Resource Management in Broadband Communication Networks

Hansen, Mads Stenhuus

Publication date:
2003

Document Version
Publisher's PDF, also known as Version of record

Resource Management in Broadband Communication Networks

Ph.D. thesis

Mads Stenhuus, M.Sc.E.E.

May 3, 2002

Resource Management in Broadband Communication Networks

Ph.D. thesis

Mads Stenhuus, M.Sc.E.E.

Research Center COM

Technical University of Denmark

May 3, 2002

Supervisors

Lars Dittmann

Ejner Nicolaisen

Andreas Magnussen (IPBlaze)

Ressourcestyring i bredbåndskommunikationsnetværk

Ph.D. afhandling

Mads Stenhuus, Civilingeniør

Research Center COM

Danmarks Tekniske Universitet

3. maj 2002

Vejledere

Lars Dittmann

Ejner Nicolaisen

Andreas Magnussen (IPBlaze)

Contents

List of Figures

List of Tables

Abstract

This thesis — *Resource Management in Broadband Communication Networks* — deals with different ways of optimizing the available resources of data- or telecommunication networks. Especially topics like optimal routing, load balancing and fast recovery of routes in case of link failures are covered.

The first part gives a brief description of some of the existing protocols for routing and controlling resources, such as RSVP, OSPF, BGP, PNNI, etc. The remaining part concerns the following two fundamentally different approaches to resource management etc.:

- Software agents
- Simulated ants

In the beginning of the part concerning software agents a description of what exactly constitutes software agents, according to the scientists and according to the organization, FIPA[1] is given. After this, the main results from the IMPACT project are presented. The IMPACT project is an EU-project that aimed at developing a demonstration platform, where software agents handled virtually all aspects of controlling an ATM-based network.

In the beginning of the part about simulated ants it is explained how scientists believe ants in nature find their way through the environment between their nest and food sources. These observations are converted into a model, which in turn leads to a Java-based ant-simulator, that can be used to thoroughly inspect the behaviour of the simulated ants. Furthermore, the model is changed in several ways, which means that the simulator is capable of investigating 10 different ant classes — some of them with improved characteristics, and others with worse characteristics (compared to the original model).

[1] FIPA: Foundation for Intelligent Physical Agents. FIPA is an organization, working to promote agent technologies and interoperability of agent systems.

The results from the investigation of the use of software agents in the context of networks is primarily *qualitative*, since it was not possible to make a lot of measurements that could in reality be compared to existing solutions. Instead, it was possible with the implemented software agent to demonstrate scenarios, where the agents could fit in additional connections by negotiations and borrowing bandwidth from each other. In addition, it was considered as an important part to build a model, that reflects the situation in a competitive telecommunication market, with several service providers sharing the same physical network.

The investigation of the simulated ants, on the other hand, gives a many promising results, suggesting that they can successfully be used to solve some problems in communication networks. For instance, the results show that a network controlled by simulated ants can balance the load quickly and efficiently, thereby postponing local hot-spots or in some cases even *avoid* hot-spots. Furthermore, the results clearly demonstrate, that systems using simulated ants obtain a virtually unprecedented robustness. A network with failing components typically return to a fully operational state within seconds or even faster.

Hopefully these results of the investigation of simulated ants — along with other results from the literature — can contribute to make the big manufacturers of telecommunication equipment consider using simulated ants in their future products.

Resumé

Denne afhandling — *Ressourcestyring i bredbåndskommunikationsnetværk* — omhandler forskellige måder, hvorpå man kan optimere udnyttelsen af de ressourcer, der er tilstede i et data- eller telekommunikationsnetværk. Herunder indgår især emner såsom optimal rutning, udjævning af belastning samt hurtig genoprettelse af forbindelser i tilfælde af kabelnedbrud.

Den første del giver en kort beskrivelse af nogle eksisterende resourcestyrings- og rutningsprotokoller, såsom RSVP, OSPF, BGP, PNNI, m.m., hvorefter den resterende del omhandler følgende to fundamentalt anderledes indgange til ressourcestyring m.v.:

- Softwareagenter
- Simulerede myrer

Afsnittet om softwareagenter lægger ud med at beskrive hvad softwareagenter helt præcist er for noget ifølge agentforskere samt ifølge organisationen FIPA[2]. Herefter gennemgås hovedresultaterne fra IMPACT-projektet, der er et EU-projekt, der sigtede imod udviklingen af en demonstrationsplatform, hvor softwareagenter forestod styringen af stort set alle aspekter vedrørende kontrollen af et ATM-baseret netværk.

I afsnittet om simulerede myrer forklares indledningsvis, hvorledes man mener, at myrer i naturen finder vej gennem deres omgivende miljø mellem deres myretue og fødekilder. Disse iagttagelser omsættes til en model, der fører til en Java-baseret myre-simulator, hvori de simulerede myrers opførsels kan iagttages grundigt. Ydermere bliver modellen ændret på en række områder, hvilket fører til at simulatoren kan undersøge 10 forskellige klasser af myrer — nogle med forbedrede egenskaber, og andre med forringede egenskaber (i forhold til den oprindelige model).

[2]FIPA: Foundation for Intelligent Physical Agents. FIPA er en organisation, der arbejder på at promovere agentteknologier, samt på at forbedre kompatibilitet mellem forskellige softwareagenter.

Resultaterne af undersøgelsen af softwareagenters anvendelse i netværkssammenhæng er først og fremmest *kvalitative*, idet det ikke var muligt at foretage ret mange målinger, der reelt var sammenlignelige med eksisterende løsninger. I stedet kunne man med de implementerede softwareagenter demonstrere scenarier, hvor agenterne kunne på plads til nogle ekstra forbindelser, ved at forhandle og låne båndbredde fra hinanden. Yderligere blev der i implementeringen lagt vægt på at opbygge en model, der afspejler situationen i et konkurrencepræget telekommukationsmarked, hvor der kan være flere serviceudbydere, der benytter det samme fysiske netværk.

Undersøgelsen af de simulerede myrer giver derimod en række lovende resultater, der antyder at de med fordel vil kunne benyttes til at løse en række problemer i kommunikationsnetværk. Resultaterne viser eksempelvis, at et netværk styret af simulerede myrer hurtigt og effektivt kan udjævne belastningen, således, at lokale overbelastningssituationer kan udskydes eller i bedste fald helt *undgås*. Ydermere demonstrerer resultaterne med al tydelighed, at man med simulerede myrer opnår en nærmest uhørt robusthed, idet defekte komponenter typisk bliver omgået i løbet af få sekunder eller endnu hurtigere.

Man kan håbe at disse resultater af undersøgelsen af simulerede myrer — sammen med andre resultater fra litteraturen — kan medvirke til at få de store producenter på telekommunikationsområdet til at overveje at benytte simulerede myrer i deres fremtidige produkter.

Acknowledgments

This Ph.D. study was financed by the Technical University of Denmark. It was carried out at the Research Center for Communications, Optics, and Materials (COM)[3] at the Technical University of Denmark (DTU).

I would like to thank Lars Dittmann, Ejner Nicolaisen, and Andreas Magnussen for their supervision — especially, I would like to thank Lars Dittmann for his never failing optimism and invaluable input.

In addition, it has been a very good experience to cooperate with Tele Danmark in the two European Projects, EXPERT and IMPACT. In particular, Poul V. Jensen (Tele Danmark) is deeply acknowledged for discussions and assistance during both the EXPERT and the IMPACT projects.

Also, I would like to thank the people from the Networks Competence Area for making sure that being at COM was always a pleasant and fun experience. In particular, I thank Anders Fosgerau of good input and Henrik Christiansen for good discussions and proof-reading.

All people participating in the IMPACT project are deeply acknowledged for two really good years of cooperation — the IMPACT project turned out to be a big success in making good friends across the European borders. In particular, John Bigham (Queen Mary and Westfield College, London) is acknowledged for good discussions about agents and simulated ants.

Lastly, my gratefulness goes to my wife, Kristine. Without her support and daily assistance this thesis would most probably never have been written.

[3] Actually, the project started at the Center for Broadband Telecommunication (CBT) at the Department of Electromagnetic Systems (EMI). However, when CBT was closed down, most of their activities were continued at COM.

Chapter 1

Introduction

Today, the Internet is taken for granted to most people as opposed to only a decade ago, where the Internet was only known to a few — mainly researchers. The development of new access technologies such as ISDN (actually, IDSN is from the 1970's), ADSL, cable modem, etc. has made it possible to offer broadband access to many private homes without having to invest a lot of money in digging down optical fibres to every home.

Some years ago, the IP-based Internet was seen by some as only a temporary solution. IP was — for a good reason — considered unsuited for offering the services guarantees required by future applications. ATM, on the other hand, seemed to solve all problems that anybody could think of, so it was for some years considered to be the protocol capable of fulfilling all needs, such as providing adequate bandwidth, quality of service guarantees, low delay (desirable for telephony), etc. However, even though ATM soon became a huge success in backbone networks, it failed in *replacing* IP, presumably due to all the existing IP-based applications and due to the fact that ATM network interface cards (NICs) and ATM switches are much more expensive than Ethernet NICs and switches. Today the IP versus ATM discussion is silenced. Instead, it is widely accepted, that the Internet will carry on using IP in the foreseeable future.

By the way, what is "broadband"? There seems to be a quite a bit of confusion about that. According to ITU-T (Recommendation I.113 [1]) the term "broadband" can be used to qualify "a service or system requiring transmission channels capable of supporting rates greater than the primary rate[1]." However, this view is not shared by everyone. For instance many service providers sell ADSL connections as "broadband", even though they are often running at speed as low as

[1]The primary rate corresponds to 1.5 Mbit/s or 2 Mbit/s depending on the geographical region.

256 kbit/s downstream and 128 kbit/s upstream. Others see broadband as something even faster than the definition by ITU-T. For instance, the Swedish parliament — "Riksdagen" — has an ambition of providing broadband access to all private homes in Sweden by the year 2005. And by broadband *they* mean at least 5 Mbit/s in both directions [2]. However, even with the definition used by many ADSL service providers, there is still a long way to go before all private homes have "broadband" connections.

It is somewhat unclear how fast connections people will actually demand in the future. Before the days of MPEG compression it was generally believed that digital video would require transmission speeds in the order of 100 Mbit/s. However, MPEG-2 (for instance used by DVD movies) has made it possible to obtain "studio quality" video at approximately 4 Mbit/s, and with MPEG-4 it is even possible to obtain acceptable quality at bit rates *below* 1 Mbit/s. This means, that once private homes are equipped with broadband connections reaching a few megabits per second (and the core network has been upgraded correspondingly), digital video can not any longer be categorized as the "killer application". However, new bandwidth hungry applications will almost certainly emerge in the future.

Even though the networks owned by the big telecommunication companies have been continuously upgraded, the Internet — as it is seen from the users' point-of-view — has not changed substantially in recent years. When will the ISPs be able to offer connections with a negotiated QoS (Quality of Service), other than best-effort? And, when will the Internet become so stable that customers will dare getting rid of their good-old telephone?

A lot of work still lies ahead before real service guarantees can be offered to Internet customers. The biggest challenge lies in the core network. In LAN environments *virtual* service guarantees can be offered using brute force — for instance by using high-speed Ethernet and switched connections. However, in the foreseeable future the resources in the core network will remain scarce, so the following topics attract great interest (the list is not exhaustive):

- Traffic engineering (e.g. load balancing)
- Network engineering
- Optimal routing
- Fault management

This thesis will start with a brief description of the resource management mechanisms happening at different time-scale. This is followed by a brush-up of the most

important protocols relevant in today's Internet, such as IP (version 4 and version 6), ATM, OSPF, BGP, PNNI, RSVP, and MPLS.

However, the main effort in this thesis has been put into the following two areas:

- Software agents
- Simulated ants

Software agents have been mentioned in widely different contexts ranging from personal digital assistants to critical applications like air traffic control. Even though software agents do not by themselves add new features, they still have many interesting things to offer. Thinking in software agents is a completely new paradigm, that can be thought of as a natural development from object-oriented programming.

The thesis will present some of the results from a European project called IMPACT, in which COM was a significant contributor (as a sub-contractor for Tele Danmark) — especially in implementing the agent called the PUA (Proxy User Agent), a complete ATM UNI-3.1 signalling stack, charging mechanisms, some example native ATM applications, and in setting up the physical test-bed that was used for the final demonstration of the IMPACT agent capabilities.

As the term "simulated ants" suggests, this approach has its roots in the collective behaviour of ants in nature. Imitating the simple mechanisms used by single ants leads to a lot of interesting results. This thesis contains a thorough investigation of simulated ants and their potential use in networks. Especially, the following topics are covered:

- The behaviour of simulated ants when used in different network topologies
- The dynamics of systems using simulated ants
- Scalability issues
- Hierarchical versus flat networks
- Load balancing
- Fault recovery

The ant simulator developed within this project is written in Java in a way that allows it to be run either as a stand-alone application or as an applet. This means that the program can be run inside a web-browser with Sun's Java-1.4 plug-in. For further information about this, see Appendix E.

1.1 Structure of the thesis

The structure of this thesis is as follows. Chapter 2 introduces different aspects of resource management and routing. In particular, the time-scales involved in the different kinds of resource management is covered.

Chapter 3 gives a brush-up of the most commonly used protocols in today's networks — i.e. IP (version 4 and version 6) and ATM. Furthermore, RSVP is briefly described, and the characteristics of some of the most commonly routing protocols such as OSPF, BGP, and PNNI are discussed.

In Chapter 4 two autonomous approaches to resource management (and more) are introduced. The first approach — *software agents* — should actually be thought of as a new paradigm, whereas the other approach — *simulated ants* — suggests specific solutions to problems such as routing, fault management, and load balancing.

The Chapters 5 and 6 describe software agents as they are seen by most agent researchers and in particular by FIPA (Chapter 6).

Chapter 7 presents some of the results obtained during the two year period of the IMPACT project. The results are mainly qualitative, since only a few quantitative results came out of the IMPACT project.

In Chapter 8 a model based on the social behaviour of ants is presented. Improvements to the model are described as well as some implementation specific details. Finally, the chapter presents a lot of results obtained using the ant simulator and some suggestions are given on how artificial ants *could* be introduced in real networks.

Finally, Chapter 9 sums up the results presented in this thesis, and this is followed by some concluding remarks.

Chapter 2

Resource Management Basics

In this chapter a general introduction to resource management in communication networks will be given. First, a broad definition of resource management will be given followed by a discussion of why it is needed. Then the mechanisms relevant in resource management that take place at the different time-scale are briefly described, and finally traffic aggregation and traffic dispersion are introduced.

2.1 What is resource management?

Generally speaking, resource management can be defined as the allocation, assignment, or provisioning of network resources (bandwidth, delay and delay variation guarantees, etc.) in order to fulfil service requests and to make sure that the quality of service (QoS) is maintained. Resource management can be either online or off-line and the two types typically cover different time-scales.

Online resource management is done in real-time (e.g. connection admission control (CAC) in ATM) and it is typically done on the basis of single connections and can therefore be very fine-grained.

Off-line resource management, on the other hand, normally deals with aggregate traffic and is therefore not as fine-grained as online resource management. The time-scales involved here can be minutes, hours, even days depending on the level at which it is done. Off-line resource management is typically about assigning resources in an optimal or near-optimal way and hereby trying to prevent future problems from occurring.

Efficient resource management is based on accurate and timely information about the condition of the network[1]. Resource management based on old information may lead to non-optimal resource allocations or even to instabilities. Accurate and timely information is particularly important for online resource management.

Also, efficient resource management depends on the *amount* of information available at the sites where resource management is actually carried out. Optimal resource allocation is only possible if *all* relevant information is available. However, too much information may also lead to excessive processing delay which, in turn, can have the same impact as resource management based on too old information (i.e. non-optimal solutions, instabilities, etc.). Another problem is that gathering information via the network (unless out-of-band signalling is used) adds extra load to the network, so in situations where resources are scarce, this may only make things worse. Normally, optimal resource management is only possible for very small (or very simple) networks. Therefore, resource management is usually based on a limited amount of information which can only lead to *near-optimal* solutions.

In addition to dealing with fulfillment of QoS requirements and maximisation of the utilization of network resources, resource management also deals with the problem of fairness among users within certain constraints and survivability mechanisms to be able to handle failures of network components and/or unexpected traffic patterns [3].

2.1.1 Routing

Even though routing may be thought of as totally different from anything that has to do with resource management, it is — nevertheless — of utmost importance. The utilization of the available network resources is highly dependent on the chosen routing strategies and algorithms.

Routing can either be static or dynamic. Static routing is adequate with some network topologies (e.g. bus and star topologies and — to some degree — ring networks) because only one route exists between two sites, so in this case little or nothing is gained from dynamic routing. However, a good set of routing tables may be good in some situations but not in other situations. In general, users are unpredictable so the network operator can never know where congestion or hotspots will occur in the future. Therefore, dynamic routing is normally preferable, since it has the ability of adapting to the ever changing traffic patterns. Actually, since telecommunication networks are dynamic and unpredictable *per se*, dynamic routing is more or less a necessity for optimal resource utilization.

[1] Link load, link/node failures, addition of links/nodes, congestion, etc.

Routing can be centralised or distributed. In centralised routing, information is gathered from all around the network and sent (via the network) to a central computer. Here the routing tables are calculated and downloaded (again via the network) to all nodes in the network. Clearly, this approach has several disadvantages:

Scalability If all routes between all nodes in a network are calculated centrally using Dijkstra's algorithm, the running time per node is $O(E \log V)$[2] [4] if the network is fully connected — i.e. the total running time is $O(EV \log V)$. Clearly, the required computing power grows rapidly with the size of the network.

Single point of failure Calculating the routing tables in only one place is vulnerable to failures. In fact, a failure of the central device does *not* necessarily mean that the network stops functioning, however, the routing tables in the nodes will not be updated in case of changing traffic patterns.

Bandwidth consumption Centralised routing requires that all relevant information is sent over the network to the central controller, and that the calculated routing tables are downloaded to the nodes. Both the information gathering and downloading of the routing tables contribute to bandwidth consumption.

If the load of a network is low, shortest path routing will normally be satisfactory. However, in general shortest path routing (e.g. fewest nodes) has some limitations. Network traffic is uneven and the traffic pattern changes over time. Furthermore, nodes on the path between two busy nodes will also be busy, so a better approach is to use *least cost* routes — with a good cost function.

2.2 Is resource management necessary?

With all the evolving network technologies an obvious thought could be that in a few years time we will all have access to bandwidth in abundance, and therefore there will be no reason to pay any particular attention to resource management. However, no-one knows what people are going to use the extra bandwidth for. Perhaps new bandwidth-hungry services will evolve, that nobody ever thought would be possible, and then bandwidth might once again become a scarce resource.

Furthermore, a network without any form of resource management could be vulnerable to malicious users. Denial of Service (DoS) attacks as well as virus and

[2] V is the number of vertices (nodes) and E is the number of edges (links). The running time stated here requires the network to be *sparse* as well as a specific implementation (the priority queue is implemented with a binary heap). Minor improvements can be achieved by using a Fibonacci heap as the priority queue [4].

worm attacks have been daily business in recent years, so perhaps resource management (and in particular policing and authentication) is now more important than ever before.

In the world of computers a famous person once claimed that “640 Kbytes ought to be enough for anybody” (Bill Gates, 1981). At that time this made perfectly sense, because most PCs had only 64 Kbytes of memory and 640 Kbytes seemed far out of reach. Nevertheless, today’s computers have a minimum of 128 Mbytes of memory and the CPUs have reached far beyond the GHz mark. Nowadays, the requirements to PCs are mainly set by the newest computer games and not by word processors and such.

So, it looks like resource management will be needed in the foreseeable future. And even if (sometime in the future) there is plenty of bandwidth, resource management will still be necessary for making QoS guarantees.

2.3 The different time-scales

Different kinds of resource management takes place at different time-scales. Figure 2.1 illustrates the time-scales involved with some classes of resource management. Of course, the exact position of the different boxes can be discussed, however, the intention is simply to give an *idea* of the time-scales.

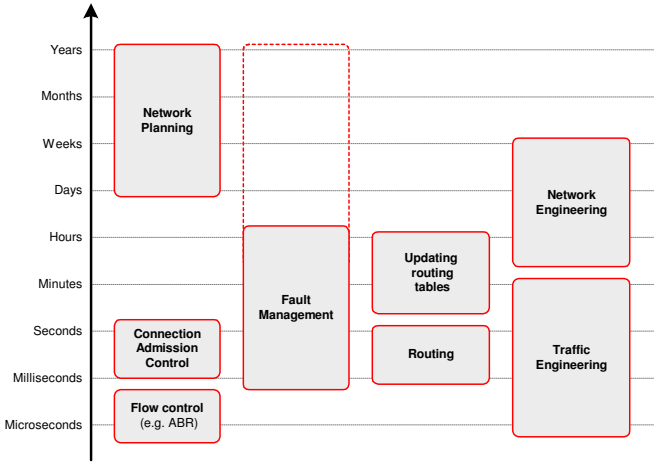


Figure 2.1 The time-scales involved in the different resource management classes.

2.3.1 Network planning

Network planning is an example of off-line resource management where the intention is to avoid future problems by putting equipment, fibres, and bandwidth in the places where the traffic is expected to be in the future. The time-scales involved in network planning may span from days to months or even years.

2.3.2 Connection admission control

Connection admission control (CAC) deals with the question of whether a new connection can be admitted so that the QoS requirements of the new connection as well as the QoS requirements of the existing connections are met. The decision of whether a connection can be made or not has to be fast (from milliseconds to a few second) in order to avoid excessive waiting times.

2.3.3 Flow control

Flow control deals with the adjustment (upwards and downwards) of the transmission speed of individual connections in order to minimise data loss and thereby minimise the need of retransmission, which only leads to increased network load. Flow control is an integral part of many network technologies like, for instance, TCP or ABR (Available Bit-Rate) in ATM [5]. Flow control has to be extremely fast and timely in order to minimise oscillations, and it typically operates in the range from microseconds to milliseconds.

The latency problem

Flow control struggles with one fundamental limitation that will always exist no matter how fast future networks become. This limitation is *latency* — the transmission speed will never be able to surpass the speed of light. If, for instance, a receiver wants the transmitter to send data at a higher speed, a certain amount of time will pass from when the transmitter sends the request for the higher speed until the transmitter receives it — and twice that amount of time will pass before the receiver starts noticing the higher speed. All in all, the information used by the transmitter to adjust transmission speed will always be outdated to some degree.

The latency problem is illustrated by Figure 2.2. In Figure 2.2a the *desired bit rate* changes slowly compared to the round-trip delay (Δt). The effect of this is that the difference between the desired bit rate and the received bit rate is relatively small. In Figure 2.2b, on the other hand, the desired bit rate changes quickly compared to Δt. As expected, this leads to a big difference between the desired bit rate and the received bit rate.

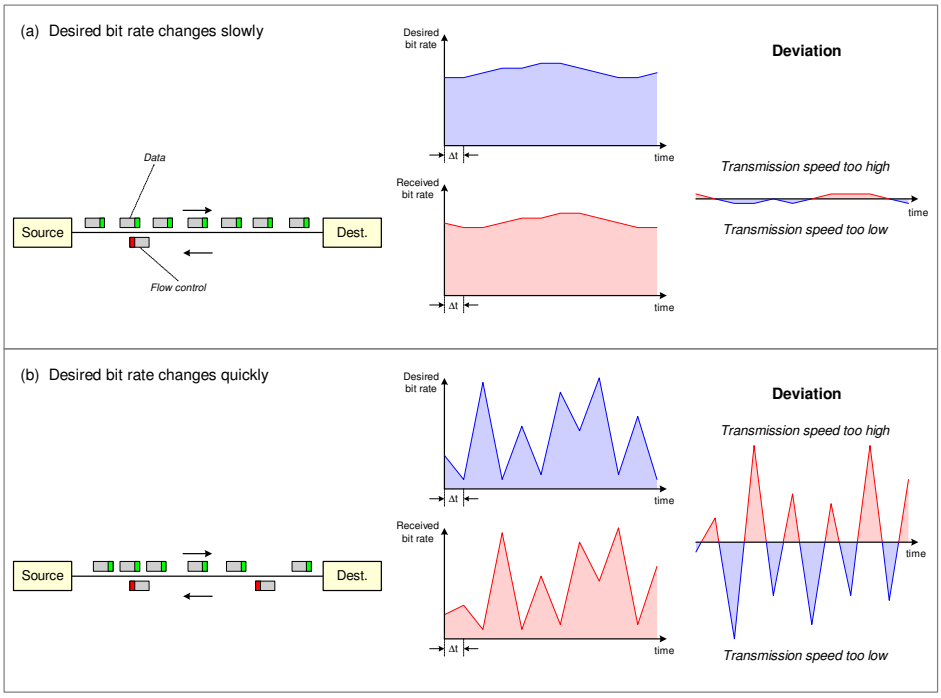


Figure 2.2 The impact of latency on flow control.

2.3.4 Fault management

The time-scales involved in fault management may vary from milliseconds to hours or, in some cases, even to years. The reason for this huge span is that the faults may either be dealt with when they occur (milliseconds to days) *or* survivability may be built into the network (weeks, months, years). In general, the task of fault management is to detect, isolate, and (if possible) repair faulty equipment in the network.

2.3.5 Routing

Routing is the process of finding a way through the network from one place to another (or between multiple places in case of multi-path routing). Routing takes place at several time-scales. The calculation of routing tables (whether it is done centralised or in the nodes makes no difference here) and download the routing tables to the node typically works in the order of minutes or hours. However, routing a specific call (connection-oriented) has to be done almost immediately (milliseconds to a few seconds) in order to avoid that the caller has to wait for too long. In connection-less networks datagrams can be sent without having to wait for a connection to be set up. However, the calculations of the routing tables used in connection-less networks are not that different from routing in connection-oriented networks.

2.3.6 Traffic engineering

The term *traffic engineering* is normally used to describe the situation where *traffic* is placed where bandwidth is available — i.e. resource allocations remain unchanged. Load-balancing belongs to this category. The time-scale of traffic engineering typically spans from milliseconds to several seconds (perhaps even a few minutes).

2.3.7 Network engineering

Network engineering goes in the opposite direction of traffic engineering. Instead of putting traffic where the resources are, the resources are put where the traffic is, or where the traffic is expected to be in the near future. The time-scale of network engineering typically spans from minutes to hours or even weeks depending on the speed of the traffic fluctuations.

2.4 Traffic aggregation

The total number of connections in backbone networks is typically so large, that the task of keeping track of all of them would be overwhelming. By aggregating traffic (or by subdividing it into a number of different classes) from several sources into fewer and higher bandwidth connections this becomes more manageable. Most network protocols — circuit-switched or packet-switched — support traffic aggregation in one form or another. Examples of these are:

ATM In ATM, several virtual connections (VCs) — a maximum of 65536, including the reserved VCs — can be "encapsulated" by one virtual path (VP).

MPLS Multi-protocol label switching (MPLS) partitions network packets into a number of so-called forwarding equivalence classes (FECs) [6]. Within an MPLS domain different packets mapped into the same FEC are indistinguishable. This means that all packets belonging to a particular FEC and travelling from a particular node will be treated equally and follow the same path.

DiffServ The differentiated services architecture, *DiffServ*, also allows for classification of packets based on the content of some portion of the packet header [7]. In this way, packets belonging to the same class are treated equally on their journey from their source to destination. DiffServ is often combined with MPLS.

SDH/SONET Synchronous digital hierarchy (SDH) [8] supports traffic aggregation through its hierarchical architecture. Low-speed connections fit into appropriate virtual containers (e.g. VC-11 or VC-12) which, in turn, fit into higher order virtual containers (e.g. VC-3 or VC-4).

2.4.1 Statistical multiplexing

When traffic from multiple bursty and independent sources is aggregated, the peak bandwidth will normally be considerably lower than the sum of the peak bandwidths of the individual connections. This leads to a *statistical multiplexing gain* for network protocols that allow sharing of bandwidth. The mechanisms of statistical multiplexing are illustrated in Figure 2.3 where three traffic sources are multiplexed together.

However, when using protocols like PDH (Plesiochronous Digital Hierarchy) and SDH there is no such effect from multiplexing different traffic sources since bandwidth can not be shared. Each circuit-switched connection consumes the same amount of resources no matter whether it is used for traffic or not.

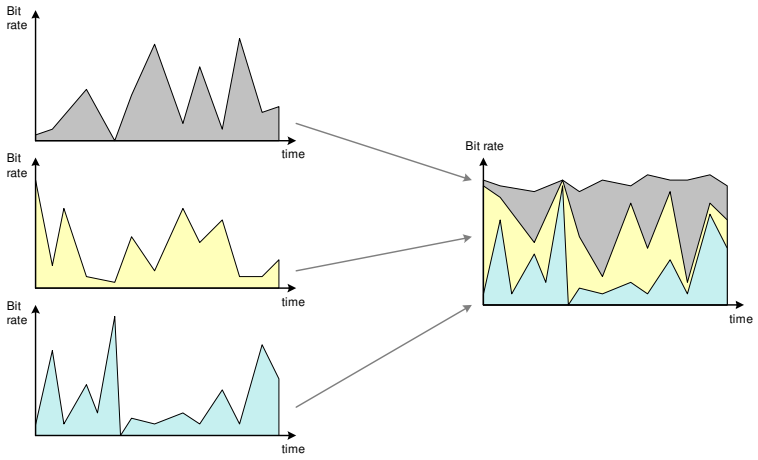


Figure 2.3 Statistical multiplexing.

Statistical multiplexing does not work well in the presence of self-similar[3] traffic sources [9]. Even if multiple self-similar and independent sources are aggregated, the aggregate network traffic will still be self-similar.

2.5 Traffic dispersion

Traffic dispersion is about spreading traffic from each source over multiple disjoint paths and transmitting it in parallel over the network [10]. Traffic dispersion can be combined with some kind of forward error correcting code that can make connections resistant to single or multiple failures, depending on the characteristics of the chosen error correcting code (see Figure 2.4).

At first glance traffic dispersion may sound like the opposite of traffic aggregation. However, this is not necessarily true, because even though traffic from a single source travels through disjoint paths, each of those paths can easily be shared by traffic from other sources.

[3]Self-similarity means that there exist non-negligible positive correlations in the traffic pattern over many time-scales, ranging from milliseconds to seconds or more [9].

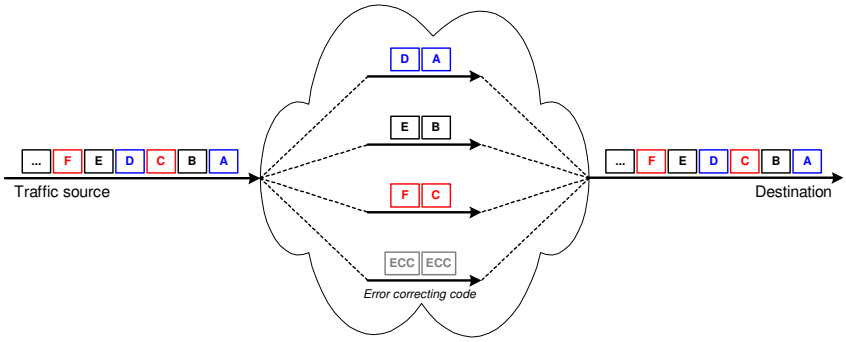


Figure 2.4 The concept behind traffic dispersion.

2.5.1 Advantages of traffic dispersion

Load balancing By routing traffic through several paths, hot-spots that could otherwise pose limitations to the utilization of the network can be avoided (or postponed). Note, however, that traffic dispersion does not by itself lead to *efficient* load balancing.

Distribution of bursty traffic Since traffic is split into a number of streams, the effect of bursty traffic will become less evident (and presumably, more manageable) with traffic dispersion.

Fault tolerance As already mentioned, traffic dispersion can provide fault tolerance when combined with some kind of forward error correcting code.

Privacy Since traffic is routed through disjoint paths eavesdropping becomes more difficult. (Of course, secure communication would still require good encryption.)

2.5.2 Disadvantages of traffic dispersion

In order to get the full advantage of traffic dispersion several disjoint paths should exist *all* the way from the source to the destination. Not only should several disjoint paths exist — the paths should also be of near-equal length (or cost) to avoid spending a lot of bandwidth just for the sake of traffic dispersion.

However, in many cases the topology (especially of the access network) does not make it possible to route traffic through disjoint paths, since most sites are hooked up to the public network through a single link (perhaps with an extra link for back-up reasons). Also, if traffic passes through several domains controlled by different network operators, effective traffic dispersion requires *several* exchange

points between any two domains. So, to make traffic dispersion a success requires major topological changes to the existing networks.

Finally, if traffic dispersion is combined with a forward error correcting code to support fault tolerance, the total amount of traffic will naturally increase, which — in turn — to some degree might neutralize some of the advantages gained through load balancing.

2.6 Summary

In this chapter some basic terms relevant for resource management have been defined. Different mechanisms have been introduced, categorized by the time-scale at which they take place. Efficient use of the available resource requires exact and *timely* information about the current network conditions (load, congestion, delay, etc.). Finally, traffic aggregation and traffic dispersion have been explained.

Chapter 3

Transmission, Resource Reservation, and Routing

As already mentioned in Chapter 2 routing and utilization of network resources are tightly connected. This chapter will give an overview of some existing technologies that are currently used for those purposes in today's Internet and other networks. In addition, multi-protocol label switching (MPLS) — a likely candidate for solving some well-known problems — will be introduced.

3.1 The Internet Protocol

3.1.1 A historical note on the Internet Protocol

The Internet Protocol (IP) has its roots in the world of data communication. It has evolved from the US military's ARPAnet of the 1960s and 1970s [11]. The idea of ARPAnet was to build a packet-switched network that could survive even though major parts of the network are destroyed. In 1974 Vint Cerf and Bob Kahn published a paper which in essence specified the design of the transmission control protocol (TCP). TCP was split into TCP and IP in 1978, where TCP would be in charge of the segmentation and the reassembly of datagrams, and IP would be in charge of transmitting each of the datagrams. Even though the Internet gained a lot of popularity in the academic world in the 1980s, the *real* breakthrough came in the 1990s with the World Wide Web (WWW) and especially in 1993 when the Mosaic web-browser became available.

3.1.2 IP version 4

The IP datagram consists of a header and a payload part as shown in Figure 3.1 [12].

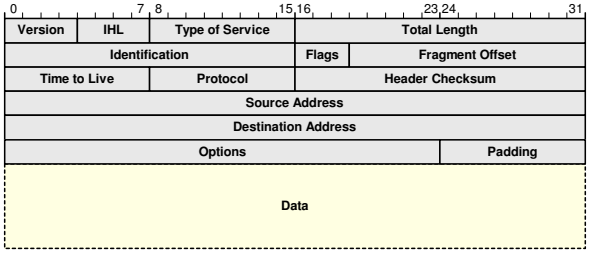


Figure 3.1 The IP (version 4) datagram.

The size of the header is 20 bytes (without options), of which the most interesting parts (for this chapter) are the four bytes source and the four bytes destination addresses. By using four bytes as the IP address there is a theoretical upper limit of $2^{32} \approx 4$ billion addresses, which seemed like a huge number back in the 1970s. However, due to reserved addresses, poor address space utilization[1] and the rapid growth of the Internet the limitations of the IP address space are becoming obvious.

Originally, the IP address space was divided into 3 classes (A, B, and C), which was later expanded to the 5 classes shown in Table 3.1 [12, 14].

Table 3.1 Original division of the IP address space.

Address Class	Address Range
Class A	1.0.0.0 – 126.255.255.255
Class B	128.0.0.0 – 191.255.255.255
Class C	192.0.0.0 – 223.255.255.255
Class D	224.0.0.0 – 239.255.255.255
Class E	240.0.0.0 – 255.255.255.255

The classes A, B, and C were intended for host addresses, class D for multicast addresses, and class E for future use. Also, routing was relatively simple, because the low number of hosts allowed for a hierarchical address structure, which in turn meant that routing could be done without looking up the *full* IP address (also called *route aggregation* [13]).

However, today the classes have been obsoleted by classless inter-domain routing (CIDR), which offers much more flexible allocations of address ranges [15]. Even

[1] For instance, MIT and AT&T were each allocated Class-A IP-addresses, so today each of them control more than 16 million addresses [13].

though CIDR still permits route aggregation at various levels of the hierarchy, there are no efficiency and scalability guarantees [13]. Nevertheless, CIDR was a necessity to expand the life span of IP (version 4).

3.1.3 IP version 6

IP version 6 (IPv6) is seen by many as the future protocol for the Internet. Today several test beds running IPv6 are in operation, and a consortium called "The IPv6 Forum" has been formed by many big companies (including leading companies such as Cisco, Nokia, and Microsoft).

The original motivation for developing a replacement for IP version 4 was the address space problem, however, NAT (network address translation) and similar techniques have provided a (temporary) solution to the lack of available IP numbers. Actually, it is claimed by some that NAT provides the *complete* solution to the IPv4 address problem, whereas IPv6-supporters claim that NAT is harmful to interoperability, since it disables true end-to-end networking [13].

Figure 3.2 shows an IPv6 datagram [16]. Compared to IPv4, the IPv6 header has been simplified and the source and destination addresses now comprise 16 bytes each, providing roughly $3.4 \cdot 10^{38}$ unique addresses. Even with a very poor utilization of the IPv6 address space there should be enough headroom for the foreseeable future. In addition, the large number of addresses makes it possible (once again) to use a hierarchical addressing scheme and thereby pave the way for simpler routing.

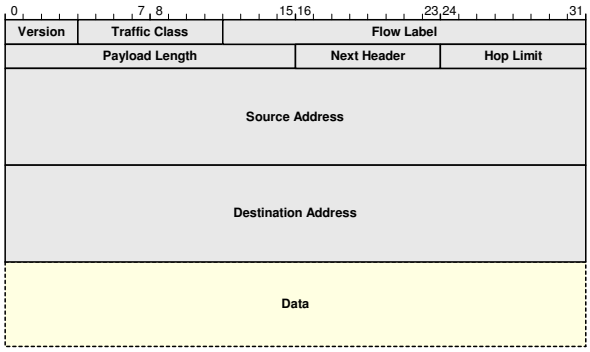


Figure 3.2 The IP version 6 datagram.

Many extra features (apart from the expanded address space) are an integral part of the IPv6 protocol suite. Some of these extra features are [13, 16]:

Auto configuration When IPv6 clients are connected to the network they are automatically assigned an IPv6 address.

Enhanced QoS capabilities The 24 bit flow label in the IPv6 datagram header enables the host to mark those packet that require special treatment by the IPv6 routers. Packets that do not require special treatment carry a flow label of zero.

Security IPv6 may provide confidentiality by encrypting only the payload or by encrypting the whole IPv6 datagram (including the header). The possible encryption algorithms are much like those used in Secure Socket Layer (SSL).

Mobility Due to the ever increasing number of hand-held devices, IPv6's support for mobility might turn out to be the most important argument for the deployment of IPv6. If an IPv6 device enters a foreign subnet it gets an extra address which has the prefix of the foreign subnet. Mobile IPv6 devices can use IPSec for all security requirements, such as data protection and authentication. In addition, IPv6 supports *route optimization* to avoid wasting bandwidth when sending data to a mobile device.

Even though all these features sound appealing they are not all unique to IPv6. Companies and standardization bodies have been quick to follow suit with extensions and add-ons for IPv4 providing more or less the same features. Hosts can be configured automatically using DHCP (Dynamic Host Configuration Protocol), SSL (Secure Socket Layer) or IPsec can provide secure connections, and even extensions providing mobility have been added to IPv4.

It will be interesting in the next years to follow IPv6's struggle to replace IPv4 — or perhaps IPv6's struggle to coexist with IPv4. In order to avoid ending up in a dead-lock situation, companies will have to *actively* support and promote IPv6 in order to make users (and ISPs) switch to IPv6.

3.2 Resource reservation protocol

The Resource ReSerVation Protocol (RSVP) working group was established by IETF (the Internet Engineering Task Force) in 1993 to specify a signalling protocol suitable for setting up reservations in IP routers to enable future real-time multimedia services [17]. RSVP does not provide any transportation service — it merely signals the QoS requirements of the end-systems to the network components.

3.2.1 RSVP basic principles

Some of RSVPs characteristics are listed in the following [18]:

- Reservation requests are passed to all network nodes in the traffic path.
- Resource reservations are initiated by the *receiving* end.
- Routers not supporting RSVP are transparent to RSVP messages.
- RSVP supports unicast and multicast traffic.
- RSVP is a *soft state* protocol, meaning that the resource reservation needs to be refreshed periodically, otherwise it is lost.
- Reservations may change during the lifetime of a connection.

The basic principle of the operation of RSVP is illustrated in Figure 3.3.

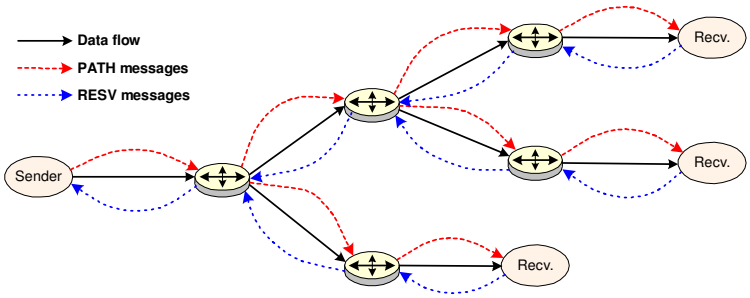


Figure 3.3 The flow of data (multicast) and reservation messages in RSVP.

The sender sends PATH messages specifying its traffic characteristics, and the receiver(s) sends back RESV messages to advertise their interest in a flow [17]. Since the reservation requests are sent from the *receiver(s)* towards the source, they merge as they meet up the multicast tree. Also, this approach allows the support for receivers with heterogeneous requirements and limitations.

Naturally, the RSVP specification also has additional message types for handling errors and for tearing down paths and releasing reservation.

3.2.2 RSVP limitations

For RSVP to be effective, all routers along the path of a datagram flow need to support RSVP. However, it *may* work with non-RSVP routers if they are not too heavily loaded, but no guarantees can be made.

The biggest problem of RSVP seems to be the scalability. Due to the high number of individual flows in backbone network, concerns have been expressed as to how many flows backbone routers will be able to support. Also, the soft-state nature of RSVP requires periodical refreshes of the resource reservations, and this might also act as a contributor to the scaling problems of RSVP.

3.3 Routing in IP networks

This section deals with routing in IP networks. The first part describes the two fundamental approaches — distance vector routing and link-state routing — to routing schemes used in the Internet, and this is followed by a comparison of the two approaches. Finally, the characteristics of the most commonly used routing protocols in today's Internet.

3.3.1 Distance vector routing

The name *distance vector routing* (DV routing) is caused by the fact that each node contains a vector with distances to all other nodes (in the domain). The term *distance* may be seen as a generalized concept. It may cover things like delay, hop-count, cost, etc. [19]

The algorithm used to generate a complete set of distance vectors is the distributed Bellman-Ford algorithm [4]:

Initialization Each node (or router) start with a vector of zero distance to itself (i.e. the networks directly attached to it), and undefined distances for all other entries in the vector.

Send All routers start advertising their current distance vector to all neighbour routers.

Receive When a router receives a new distance vector it updates its own distance vector by comparing the current distances to the distances in the received vector added to the *distance* of the link the distance vector arrived at. When finished, the router goes to **Send**.

An example showing the mechanisms of the distributed Bellman-Ford algorithm is shown in Figure 3.4[2]:

- In step 1 all routers A, B, C, D, E, and F only know the distance to themselves (which is zero).
- In step 2 router E advertises its DV to A, B, C, D, and F. Each of these routers compare the current values in their DVs to the sum of the received DV and the *distance* of the link the DV arrived at. The distance vector is the updated with the lowest number.
- This continues step 3 through 11 — the order of the advertisements is chosen more or less randomly.
- In step 12 the DV in routers show the shortest distance to all other routers.

Claiming that routers only contain a single distance vector is too simplified. Actually, each router contains the last received distance vector for *each* of its neighbours added to the distance to those neighbours. These distance vectors effectively make up the routing table each router. The routing tables for the each of the nodes from the previous example are shown in Figure 3.5 where the shortest distances to each destination node is marked with the colour red. The values marked with red are, in fact, the *distance vector* of each of the routers. For simplicity reasons, the routing tables of the routers have not been shown in Figure 3.4.

Link failures in DV routing

In DV routing, the distance vectors are periodically advertised even after steady-state has been reached. This is necessary to cope with changes in the network such as changing distance between two nodes or link failures. If, for instance, the link between router D and router E fails, the routing tables change as shown in Figure 3.6.

As a result of the link failure, the distance vectors that both routers will advertise from now on have changed. After a while these changes will have reached all routers in the network, causing them all to avoid the failed link.

Routing loops in DV routing

When using distance vector routing, temporary routing loops *may* occur in case of link failure(s). This actually happens in this example if the link between D and F

[2]The example is shown with one update at a time. However, in reality the updates are made in parallel throughout the network.

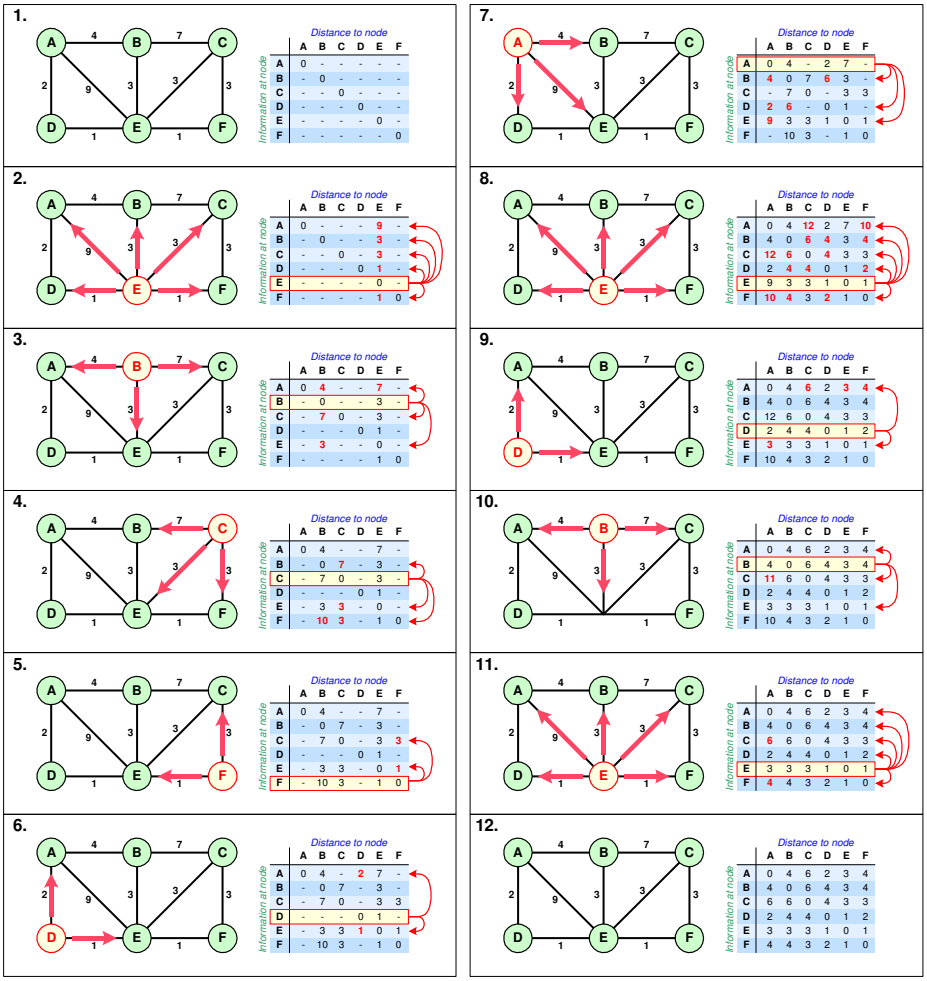


Figure 3.4 Distance vector exchange in distance vector routing.

Routing table of router A

Destination \ Next hop	B	D	E
B	4	6	12
C	10	6	12
D	8	2	10
E	7	3	9
F	8	4	10

Routing table of router B

Destination \ Next hop	B	D	E
A	4	13	6
C	10	7	6
D	6	11	4
E	7	10	3
F	8	10	4

Routing table of router C

Destination \ Next hop	B	E	F
A	11	6	7
B	8	6	7
D	11	4	5
E	10	3	4
F	11	4	3

Routing table of router D

Destination \ Next hop	A	E
A	2	4
B	6	4
C	8	4
E	5	1
F	6	2

Routing table of router E

Destination \ Next hop	A	B	C	D	F
A	9	7	9	3	5
B	13	3	9	5	5
C	15	9	3	5	4
D	11	7	7	1	3
F	13	7	6	3	1

Routing table of router F

Destination \ Next hop	C	E
A	9	4
B	9	4
C	3	4
D	7	2
E	6	1

Figure 3.5 The routing tables of each of the routers.

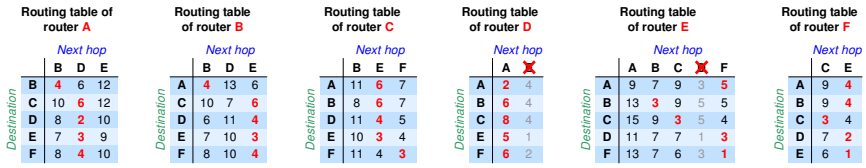


Figure 3.6 The routing tables of the routers immediately after the link between D and E has failed.

fails. Based on the routing tables in Figure 3.6, router A will still choose router B as the next hop, if A wants to route an IP datagram to — for instance — router E. Router B, on the other hand, returns the datagram to router A, and the loop is established. Of course the loop will vanish, when the distance vectors have been passed around for long enough, and the system has reached steady-state.

There are methods for avoiding routing loops in DV routing. For instance, if each DV advertisement carries the entire path, a router can reject a route if it sees itself in the path.

3.3.2 Link state routing

Link state routing (LS routing) is fundamentally different from distance vector routing in the way information is passed around in the network *and* in the way the preferred routes are found. In link state routing each router is assumed to know the state (i.e. the cost) of each of the links to its neighbours. These "link states" are then broadcast to all other nodes, and in this way each node gets a *global* view of the network (as opposed to DV routing, where the nodes only know their neighbours). In LS routing the best paths are computed *locally* in each node using Dijkstra's algorithm[3] [4, 20].

Link failures and routing loops in LS routing

When a link failure is detected, the adjacent nodes broadcast information about the failing link to all other nodes in the network. The failed link will be effectively avoided once all node have run Dijkstra's algorithm again with the updated link states.

[3]No example will be shown here, since the only interesting thing in LS routing is that the shortest-paths are found in *some* way. In DV routing, on the other hand, the distributed Bellman-Ford algorithm may *by itself* cause routing loop problems. This is *not* the case in LS routing.

However, it may last a while before all routers have calculated a new set of routing tables, and in the mean time (transient) routing loops may occur, which leads to loss of data and to highly loaded links.

3.3.3 Distance vector routing versus link state routing

In DV routing each node send everything it knows to all its neighbour nodes. In LS routing, on the other hand, only the adjacent link states are sent, but in this case they are sent to *all* nodes in the network. Due to this, the size of the messages sent in DV routing may become much larger than messages sent in LS routing. However, the potentially large routing messages in DV routing does not necessarily become a problem, since they are only sent over short distances.

The convergence speed of LS routing can be very fast, once the router gets an overview of the entire network. The distributed nature of the algorithm used in DV routing may limit the convergence speed of DV routing, unless the routers are forced to use fast updates.

One of the biggest *potential* problems of DV routing is that bogus information from one node spreads out to all nodes (through the distributed Bellman-Ford algorithm), which may case stability problems. In LS routing, however, incorrect information will only lead to localized problems.

Finally, since routers using LS routing need a global overview of the entire network topology, their space requirement may become very large. In DS routing, on the other hand, the space requirement is limited by the fact that only the states of the *neighbour* nodes are known.

It is difficult to announce a clear winner among the two routing candidates. DV routing has been used successfully for many years (and it is still widely deployed), but nevertheless LS routing is gaining popularity.

3.3.4 Routing information protocol

The Routing Information Protocol (RIP) [19, 21] is one of the most enduring of all routing protocols [22]. RIP is a distance vector routing protocol (see Section 3.3.1) that works well in moderately sized networks.

Due to the RIP packet format, the maximum diameter of the network using RIP is limited to 15. This means that once a packet reaches hop number 16 it is discarded. Actually, this limitation is — at the same time — a disadvantage *and* an advantage. The 15 hop limitation prevents RIP from being used in very big networks, but it *also* prevents routing loops from continuing indefinitely [22].

Holddown

RIP contains more stability mechanisms than just the hop-count limitation. One of these mechanisms is called *holddown* [22]. Holddown simply forces routers to *delay* the propagation of information under certain conditions — e.g. link failure of suddenly changing metrics. In this way, the neighbour of the "silent" router will think that it is dead, which — in turn — will force them to find alternative routes by themselves.

In the example shown in Figure 3.6, holddown will make router A believe that the routers B and E are dead. This will prevent routing loops between router A and router D[4].

The downside of the holddown mechanism is that by delaying information the convergence time of the distributed Bellman-Ford algorithm will increase.

RIP version 2

The original version of RIP [19] did not support any mechanisms for preventing routers from spreading bogus information through the network. This problem has been addressed in RIP version 2 [21], by including some simple authentication mechanisms [22].

3.3.5 Open shortest-path first

As the name *open shortest-path first* (or simply OSPF) suggests the protocol is open — i.e. its specification is in the public domain [22, 23]. Furthermore, OSPF selects routes using "shortest-path first" discovered with the help of Dijkstra's algorithm. OSPF is a link state routing protocol (see Section 3.3.2) which means that each router has the full overview of the network.

OSPF is an interior gateway routing protocol that can operate in a hierarchy (contrary to RIP) and the largest entity in the hierarchy is the *autonomous system* (AS). An autonomous system is a collection of networks with a common administrator all sharing a common routing strategy [22].

An AS can be divided into several areas, and the topology of each area is invisible outside the area. Dividing a network into smaller areas results in a lower load from the messages passed around by the OSPF protocol.

Figure 3.7 shows an autonomous system consisting of 14 routers and three areas. The six routers I, J, K, L, M, and N make up the backbone (the backbone is in

[4]Of course, it may last a while before router A take notice of router D's silence. During this period routing loops may still occur.

itself an OSPF area at a higher level), which is responsible for sending routing informations between the areas.

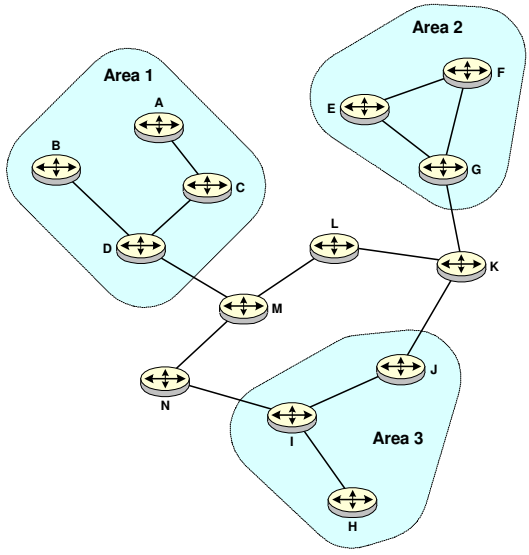


Figure 3.7 An OSPF autonomous system (AS) with three areas.

Additional features in OSPF

OSPF includes features not found in RIP, making it the preferred choice as the routing protocol used inside autonomous systems. For instance, if multiple equal-cost paths exist to a destination, OSPF makes sure that datagrams to that destination use both routes (load balancing). Also, OSPF supports priority routing based on the TOS (Type of Service) field in the header of IP packets [22].

3.3.6 Border gateway protocol

The border gateway protocol (BGP) is an *exterior gateway routing protocol* — also referred to as an *interautonomous system routing protocol* [22]. Actually, BGP may be used inside an AS — in that case it is referred to as Interior BGP (IBGP) — as well as *between* autonomous systems (External BGP, EBGP). However, IBGP will be silently ignored here. Figure 3.8 shows three autonomous systems (running OSPF or another interior gateway routing protocol internally) connected through four border gateway routers.

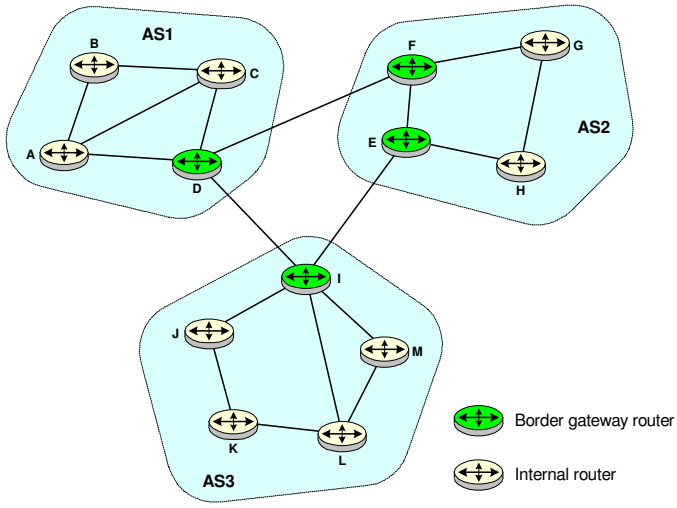


Figure 3.8 Three autonomous systems exchanging routing information using BGP.

BGP is a distance vector routing protocol[5] (see Section 3.3.1) meaning that each border gateway send their current distance vector to their neighbours and use the distributed Bellman-Ford algorithm to find the preferred routes [24]. Unlike RIP, BGP does not use the simple *holddown* mechanism to limit the damage from routing loops. Instead routing loops are completely avoided by not only sending distance vectors to their neighbours, but also the routes leading to those distances. In this way a router running BGP can avoid sending packets along routes including the router itself, and this effectively avoids loops (due to this, BGP is often referred to as a *path vector protocol*). In addition, BGP uses TCP for distributing data between routers in order to ensure data delivery (in RIP and OSPF data is sent over raw IP).

Of course, the price of the loop-prevention mechanism used in BGP is that larger amounts of information are passed around among border gateway routers, which may lead to a significant load on the links between them.

Studies have suggested that BGP has some instability problems in its current implementation [25, 26]. So-called *route flaps*[6] have been observed, that may affect the network performance due to excessive processing load (within the routers), high load of the links between the routers, slow network convergence etc. [26]

[5]The routing protocol used in BGP is often referred to as a "path vector" protocol.

[6]Quickly changing paths between two end-points using different transit networks.

3.4 Asynchronous transfer mode

3.4.1 Introduction

The asynchronous transfer mode (ATM) has its roots in the world of telecommunication. Originally, the intention was that ATM should bridge the gap between datacommunication and telecommunication, by combining the best parts of the two worlds, and in the long run perhaps replace (almost) all other competing protocols.

ATM has indeed been a huge success even though it is invisible to the major part of the population. For instance, backbone IP networks typically run on top of ATM, and with the introduction of xDSL[7] to resident users ATM has even made its entrance in many private homes. But, the widespread availability of native ATM end-to-end connectivity is still missing to make the picture complete — and with the advances in IP routing and with MPLS, the original dream of one united network using ATM will most likely never become a reality.

3.4.2 Cells, switching, and signalling

ATM is a circuit-switched protocol using small fixed-sized packets (cells) with a length of 53 bytes — 5 bytes for the header and 48 bytes for the payload. Figure 3.9 shows the format of ATM cells [27]. ATM cells at the User-Network Interface (UNI) are slightly different from ATM cells at the Network-Network Interface (NNI) — at the NNI the VPI field is 12 bits long and only 8 bits long at the UNI.

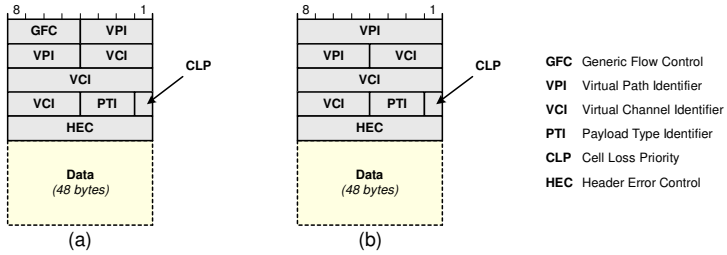


Figure 3.9 The format of an ATM cell at the UNI, (a), and at the NNI, (b).

In ATM, the switches choose the path of ATM cells based on their VPI/VCI values. At each intermediate node the VPI/VCI values are examined, the output port for the cell is found by table look-up, and (possibly) new values are written into the VPI/VCI fields of the cell. The VPI/VCI values of ATM cells only have local

[7] xDSL: Digital Subscriber Line (ADSL, SDSL, etc.)

significance — i.e. they are negotiated *before* the ATM connection is ready for use. ATM makes a distinction between virtual paths (VPs) and virtual connections (VCs). For virtual paths only the VPI values of the cells are examined at intermediate nodes, and for virtual connections both the VPI and the VCI are examined. In this way, one VP may contain a maximum of 65,536 VCs.

The nodes where ATM cells are examined (and their header processed) are called switches or cross-connects. The distinction between switch and cross-connect can be made by the supported methods of establishing connections through that particular switch/cross-connect. ATM switches can set up connections using signalling (through the control-plane) *or* directly through the management interface, whereas cross-connects must be configured through the management interface, since they lack the support of signalling [1].

In ATM networks connections can either be established manually or on request using ATM signalling. Several signalling specifications exist from ITU-T and from the ATM Forum — e.g. Q.2931 [28] and UNI-4.0 [29]. Q.2931 only supports point-to-point connections, whereas UNI-4.0 supports multicast and even so-called *leaf-initiated join* — i.e. receiver may hook themselves to already established multicast trees.

3.4.3 Resource reservation in ATM

In general, resources in ATM networks are set up at the same time as a connection is set up — and if the requirements for a connection can not be fulfilled the connection is rejected by the CAC (Call Admission Control) mechanisms, and if some switches have made temporary resources reservations, they are cleared. In contrast to RSVP, the resources remain untouched until the connection is *explicitly* torn down — i.e. resource reservation in ATM switches make of use hard states.

Normally, ATM switches offer a number of different ATM services classes. The service categories defined by the ATM Forum and ITU-T are listed in Figure 3.10 [3]. Clearly, ATM Forum's CBR (Constant Bit-Rate), rt-VBR (real-time Variable Bit-Rate), and ABR (Available Bit-Rate) services classes correspond to the ITU-T service classes DBR (Deterministic Bit-Rate), SBR (Statistical Bit-Rate), and ABR, respectively. The meaning of the different parameters will not be covered here. There are excellent descriptions in the literature, for instance pp. 9–13 in [3].

The only ATM service class requiring (explicit-rate and hop-by-hop) flow-control is ABR. ABR was a promising service class first proposed by the ATM Forum, that should enable near-optimal resource utilization in ATM networks. However, the deployment and usage of ABR has been very limited.

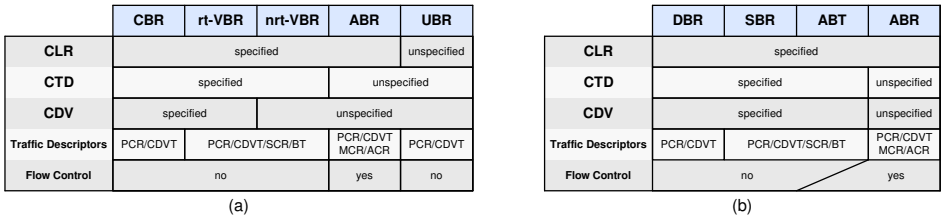


Figure 3.10 ATM layer service categories specified by the ATM Forum, (a), and by ITU-T, (b).

Figure 3.11 illustrates the control functions acting on cells travelling from the source to their destination [3]. When cells leave their source they may pass through a *shaper*[8] before they enter the network domain. As the name suggests, the job of the shaper is to shape the traffic flow to make sure that the user stays within the envelope that was specified at connection setup — e.g. temporary bursts may be shaped into a burst with a lower PCR (and a longer duration).

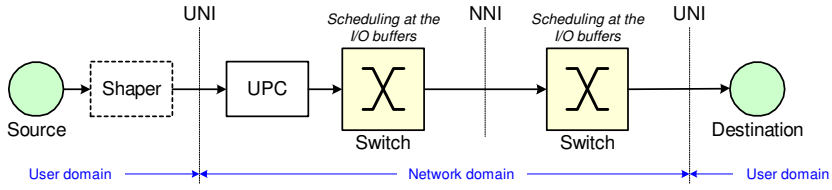


Figure 3.11 Control functions performed on cells travelling in one direction.

Once the ATM cells enter the network domain, they are *policed* — i.e. this time the *network* makes sure, that the user stays within the envelope by using UPC (Usage Parameter Control). Normally, the UPC is using a simple dual leaky-bucket mechanism, to prevent too long bursts and to prevent too high sustained cell-rate as illustrated in Figure 3.12 [30].

The two virtual buckets exist for *each* policed stream of cells, and for each incoming ATM cell the fillings of both buckets belonging to that stream increase by a small amount. In addition, the fillings of both buckets is reduced with constant rates, Rate 1 and Rate 2. If one of the buckets overflow, the ATM cell is discarded. In the figure, the small bucket, Figure 3.12(a), polices the peak cell rate (PCR), whereas the large bucket, Figure 3.12(b), polices the sustained cell rate (SCR).

[8]The shaper is not mandatory, however it is highly recommendable, since bursty traffic is difficult to cope with in the network.

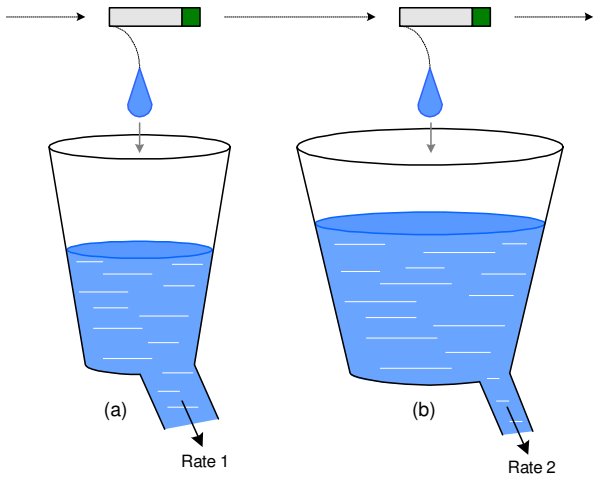


Figure 3.12 The leaky bucket principle (Rate 1 > Rate 2).

In general, ITU-T and the ATM Forum have specifications for how to signalling information among network nodes and end systems. However, all the specifications include a top-level process called the *application* process, which is not part of any standardization [30]. The application process is responsible for virtually everything that takes place at a higher level — for instance it decides whether a switch can accept the requested or not (CAC). This means that the actual implementation of CAC in ATM switches is a competitive feature.

3.4.4 Routing in ATM networks

Routing in ATM has not attracted nearly the same attention as routing in IP networks. The reason for this could be that ATM networks can be used *without* routing protocols, since the most common use of ATM has been (and still is) to manually set up semi-permanent point-to-point connections through the management interface of the relevant ATM switches.

IP networks, on the other hand, *need* routing in order to work[9] and this may have served as an important motivation for developing the rather large number of routing protocols for IP traffic.

[9]However, IP routing is not required in networks providing full logical IP connectivity between all hosts — e.g. Ethernet or Token-Ring LANs.

Interim inter-switch signaling protocol

The interim inter-switch signaling protocol (IISP) is basically equal to ATM UNI signalling with a few simple extensions to support signalling between ATM switches (NNI signalling) [31]. As the name *interim* suggests, IISP was only intended as a temporary NNI signalling solution until something better had been developed.

IISP works like this: A user sends a connection request to an edge switch using one of many UNI signalling protocols. The edge switch then imitates the user, and asks the next switch along the chosen route to set up the requested connection. This process is repeated until the destination is reached *or* until the request fails, for instance, due to lack of network resources.

While the connection request moves through the ATM network (hop-by-hop) each switch acts both as the user-side and as the network-side of the used UNI signalling protocol stack.

Routing in IISP

Routing in IISP is extremely simple and actually not particularly interesting. A fixed routing algorithm with manually configured static routes is used [31], which means that in case of link failure there are no alternative routes. An excellent IISP routing example is shown in Appendix A in [31].

Private network-to-network interface

The acronym PNNI can either stand for *Private Network-to-Network Interface* or *Private Network Node Interface*. The two meanings of PNNI reflect the two possible uses [32]:

1. A protocol is defined for *signalling* user requests to the network for setting up point-to-point and point-to-multipoint connections across the ATM network. PNNI signalling is based on UNI signalling and extended with some additional mechanisms.
2. Another protocol is defined for distributing topology information among switches, which is used to compute paths. PNNI supports auto-configuration of the PNNI hierarchy based on the address structure.

Routing in PNNI

PNNI routing based on the link state routing technique (see Section 3.3.2), where each node calculates the shortest path based on a global topology overview stored

locally. Each node sends routing information periodically and every time a state change occurs.

PNNI networks are hierarchical, which means that each switch has to contain limited information about the network. The network is divided into so-called *peer groups*, which are a groups of nodes at the same level in the hierarchy. Each peer group appoints a peer group *leader* (based on its ATM address) with the role of "representing" the entire peer group at a higher level in the hierarchy. At higher levels, peer groups are formed containing only peer group leaders, and they also appoint a leader among themselves representing them all at the higher level. This continues until the top-level is reached[10] [32].

Topology information is only sent to all other nodes *within* a peer group. The leader then "summarizes" the information and sends it to other peer group leaders at a higher level.

Figure 3.13 shows how an ATM network using PNNI arranges itself into a hierarchy with three levels. The local topology views of three nodes in the network are shown in Figure 3.14. Clearly, the hierarchical structure simplifies the process of calculating routes.

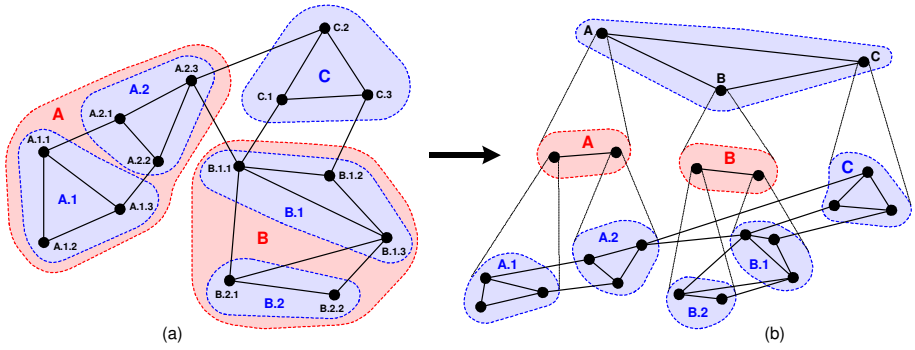


Figure 3.13 ATM network using PNNI (a) organized in a hierarchy (b).

Apart from simply finding routes through the network, PNNI also has to make sure that a path for a virtual connection is selected in such a way the links and switches along the path appear to be capable of fulfilling the QoS and bandwidth requirements of the VC [32]. For instance, if a switch along the route can not fulfil the requirements, PNNI moves back to the previous switch and tries to find an alternative route from there (this is called *crankback*).

[10] PNNI supports a huge number of hierarchy levels (up to 104 levels), which makes sure that it scales well, even to large world-wide networks.

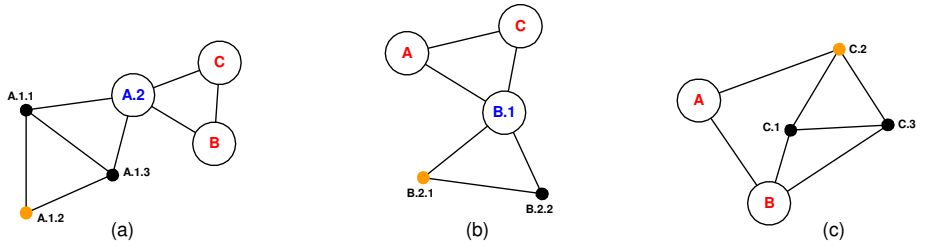


Figure 3.14 Local view from three nodes: (a) A.1.2, (b) B.2.1, (c) C.2

3.5 Multi-protocol label switching

Multi-protocol label switching (MPLS) is not really a protocol in the way protocols are normally thought of. It is more like a *concept* for how to build highly scalable (IP) backbone networks by making heavily use of traffic aggregation to reduce the number of simultaneous flows. In fact, it can be seen as a shim-layer between level 2 and level 3 in the 7-layer OSI protocol model (i.e. between the data link layer and the network layer) [6, 33].

The main idea behind MPLS is to do all complicated things at the edge of the network (where the number of flows is manageable) thereby keeping the functionalities in the core part of the network as simple as possible. The number of flows in the core network is reduced considerably by making heavy use of traffic aggregation into so-called forward equivalence classes (see Section 3.5.3). Simply put, an MPLS network can be seen as one huge distributed IP router.

3.5.1 The situation before MPLS

In the mid-90s the most pragmatic solution for building high-performance IP backbone networks was to use ATM as the underlying technology. However, IP and ATM remained as two logically decoupled networks, where ATM was merely used to provide data link layer connectivity [34]. IP packets still had to be handled by the relatively slow IP routers, so there was not much to be gained from the faster ATM switches.

An example clearly illustrating the lack in integration between the IP and the ATM layer, is the following statement (translated from Danish):

> *"We at Tele Danmark don't have an ATM network anymore — now everything runs over IP."*

This statement from a Tele Danmark employee is from an IDA (The Society of Danish Engineers) seminar with the title "Convergence" held in March 1999. The truth, however, was that their IP network ran on top of ATM, but apparently he did not know that!

3.5.2 Edge and core label switched routers

The fundamental actors in MPLS networks are the label switched routers (LSRs). LSRs exist in two categories [33]:

Edge LSRs are high performance packet classifiers located at the edge of the MPLS network that can apply labels to packets entering the MPLS domain.

Core LSRs are — as the name suggests — located in the core of an MPLS network. Core LSRs must be able to process the labeled packets extremely fast.

Figure 3.15 shows a simple MPLS network example.

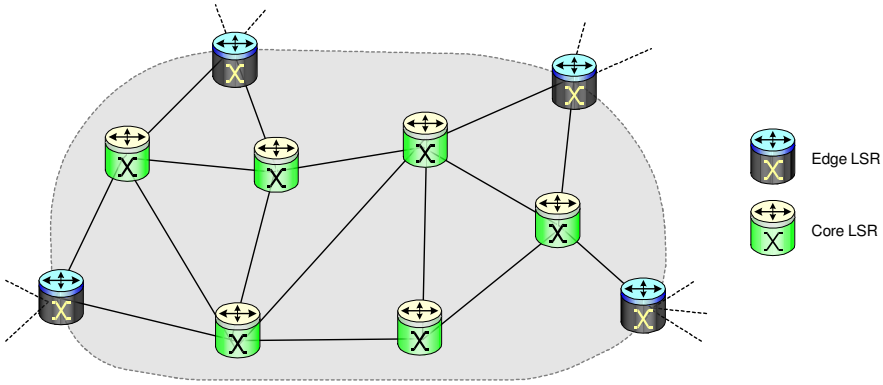


Figure 3.15 MPLS network example with edge and core label switched routers.

When IP packets arrive at an edge LSR, they get a label that can be used by other LSRs to forward the IP packets through the MPLS network until they reach their egress node (also an edge LSR), where the label is removed before they are forwarded using traditional IP routing techniques. In this way, the core LSRs avoid the slow process of looking up the IP address.

Paths through an MPLS network are referred to as *Label Switched Paths* (LSPs), due to the way packets are forwarded. Of course, the LSRs need tables to translate

the labels of incoming packets to an output port (and possibly a new label). The information for these tables is distributed using the Label Distribution Protocol (LDP) among others.

3.5.3 Forward equivalence classes and QoS

The number of flows in a backbone network is not reduced by simply translating IP addresses into unique labels, when IP packets enter an MPLS domain. However, MLPS performs traffic aggregation by categorizing incoming packets into forwarding equivalence classes (FECs), where packets belonging to a particular FEC will all get the same treatment[11] in their way through the MPLS network [35].

This effectively reduces the number of flows within the MPLS network, and it also enables support for QoS. However, the QoS support is limited, since all packets belonging to a single FEC gets the same treatment — i.e. the MPLS network can only guarantee an *overall* QoS for the packets in a FEC and not ensure equal treatment for all sources.

3.5.4 MPLS and ATM

Even though MPLS does not specify what protocol to use at the data link layer, ATM can be seen as a well-suited candidate with proper software updates in the switches. The VCI/VPI fields in the ATM cell header can contain the label, and LSPs (virtual connections in the ATM world) can be set up using ATM signalling.

However, ATM does not support all MPLS features. For instance, ATM does not support *label stacks* or *label merging*. With label stacks, each LSR pops a label of the stack and uses it to determine the outgoing port — i.e. the packets are effectively source-routed [6].

3.6 Summary

This chapter has given an overview of some of the most common protocols used in today's Internet (and networks in general), IP and ATM, and described a likely candidate, MPLS, for solving some of the scalability problems and enabling (limited) QoS support. MPLS can be seen as a competitor to ATM, but also as a way of integrating IP and ATM, so the future of ATM looks rather good.

In addition, the principle of the most commonly used routing mechanisms (distance vector routing and link state routing) have been explained, and their use

[11] This is not entirely true when DiffServ is used together with MPLS. With DiffServ, packets within one FEC may indeed be treated differently.

in IP routing as well as PNNI routing has been addressed. Of the two principal routing mechanisms no clear winner can be announced. Although the algorithm used in distance vector routing (distributed Bellman-Ford) may have some potential problems with route loops, they can be suppressed using simple extensions (e.g. *holddown* in RIP) or completely avoided by adding the complete routes to the routing information that is exchanged between the routers (this is done in BGP).

Finally, the chapter has described how resource reservation is done (or *can* be done) in IP and ATM based networks. Table 3.2 outlines some of the main characteristics of resource reservation in IP networks and resource reservation in ATM networks.

Table 3.2 Resource reservation in IP and ATM networks.

Description	IP + RSVP	ATM + Signalling
Setup + Reservation*	Sequential	Simultaneous
Multicast support	Yes	Yes (with UNI-4.0)
Reservation origin	Receiver	Sender (root)
Reservation changes	Yes	No†
Reservation state	Soft	Hard
Routing protocols	RIP, OSPF, BGP, ...	PNNI, IISP
Static routing	Yes	Yes (IISP)
DV routing protocols	RIP, BGP	—
LS routing protocols	OSPF	PNNI
Route changes	Time-out	Re-establish connection

* The order in which connections are established and resource reservations are made.

† Actually, the eight ITU-T recommendations Q.2963.*X*, add limited support for modification of traffic parameters for established connections by the connection owner.

Chapter 4

Two Approaches to Distributed Resource Management

This chapter introduces some approaches to distributed (and autonomous) resource management. The first approach is using software agents for dealing with tasks that would otherwise be dealt with by the control or management plane, by central computers — or even by humans. The second approach is imitating the behaviour of simple biological creatures (in this case: *ants*) and investigating how that can be taken advantage of in communication networks.

4.1 The rationale

Todays telecommunication networks are complex and highly heterogeneous. The mechanisms used to control and manage these networks make heavily use of algorithms (especially Dijkstra, distributed Bellman-Ford, and other algorithms from graph theory) that can be mathematically proven to work. These mechanisms work well under static conditions, however some of them react slowly or require much processing power in highly dynamic networks. Solutions that may be optimal in some situations may suffer heavily in other situations.

Also, due to "poor" scalability many protocols become too slow when the size of the network reaches a certain limit (see, for instance, Section 2.1.1). The solution to this problem has traditionally been to use a hierarchical approach with many independent entities at the lower layers and fewer entities at the higher layers. This price of this subdivision is that each entity can only work towards a *local* optimization of the available resources — and the sum of all locally best solutions will normally not be equal to the *globally* best solution.

Another simple and popular solution to reduce the complexity of networks has been to use simple topologies that can very easily be understood and managed (e.g. tree and ring topologies). However, with these topologies there are very limited possibilities for implementing survivability techniques and they have virtually no possibility of adapting to changing traffic patterns.

4.2 Software agents

Software agents have gained a lot of attention over the last years. They have been mentioned in widely different contexts, such as air traffic control, personal digital assistants, information retrieval, electronic commerce, etc. [36–38] However, in recent years the use of software agents in telecommunication networks has also gained attention [39]. Recently, the IMPACT project built a demonstration system that used intelligent agents to control and manage an ATM network (see Chapter 7), and currently another European project in the IST program called SHUFFLE[1] is investigating the use of intelligent agents for controlling the resources in UMTS networks.

Using software agents does not add a wide range of extra properties and features *per se* — it shall be seen more like a new *paradigm*. Splitting tasks into subtasks that can be handled by self-contained and independently executing software entities (agents), *does* potentially add a lot flexibility compared to traditional procedural and object-oriented programming, with only one executing thread. Furthermore, by using special agent programming toolkits, the developer gets many features for free, like — for instance — communication mechanisms, life cycle management, yellow-/white-pages services etc. (see Section 5.3.4).

One of the key selling points of software agents is *autonomy* — i.e. an agent must (or at least: *should*) be able to work with limited or even without human intervention. However, autonomy might turn out to be a good selling point as well as the biggest nightmare of the network owners — so before agents can become a reality in real telecommunication networks, they must prove themselves as robust and trustworthy.

Formal definitions of software agents as they are defined by most agent researchers and by the standardization initiative called FIPA, are given in the Chapters 5 and 6.

[1] SHUFFLE web pages: `http://www.ist-shuffle.org/`

4.3 Ants

This section explains the trail laying properties of social behaving insects such as ants. In the literature, ants are sometimes referred to as *agents* [40, 41], however using the term agents to describe simple creatures like ants does not go well in hand with the definition of agents used by most researchers. So, in the following, the terms "simulated ants" and "artificial ants" will be used interchangeably to describe the individual entities with ant-like behaviour or to describe the whole system using the collective behaviour of ants.

Individual ants are believed to be very unsophisticated and to have a very limited memory. Moreover, the movement of a single ant may seem to be quite random. However, ants are social insects, and in spite of the simplicity of a single ant, studies of the behaviour of ant colonies (with thousands or millions of ants) have revealed some interesting properties such as [40, 42]:

- Ants can regulate the nest temperature very accurately (within 1°C).
- Ants can co-operate in carrying large objects.
- Ants can find the shortest or near-shortest trails from their nest to food sources.

Naturally, the last property is of great interest to communication networks. Especially, one might expect that systems based on ant-like behaviour will be *very* robust, since ant colonies in nature are not dependent on the individual, but instead on the collective behaviour. Furthermore, since nature has evolved over millions of years, the mechanisms used by ants have indeed been tested *thoroughly* in the nature's own testbed — and with great success.

4.3.1 The behaviour of ants in nature

When ants in nature communicate, they often do so using indirect communication through their environment — this is called *stigmergy* [42]. When ants move around in their environment, they deposit *pheromones*[2] wherever they go. These pheromones are used by the other ants, when they determine the direction — trails with high concentrations of pheromones, will be selected more frequently than trails with low concentrations of pheromones.

A very simple example, hinting the collecting behaviour of ants is illustrated in the Figures 4.1, 4.2, 4.3, and 4.4. Figure 4.1 shows a system consisting of one ant's

[2] A chemical substance, with a certain scent, to which other ants react.

nest (with two eager ants), one food source, and two trails between the nest and the food source. The two trails have not been used before, so the two ants can not tell any difference between them — i.e. there is a 50-50 chance of going either way.

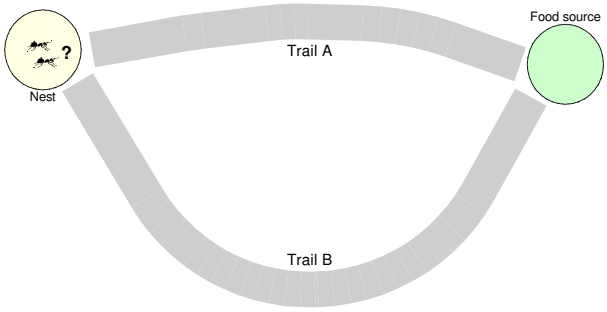


Figure 4.1 The initial situation.

At a given moment the two ants decide to leave their nest to go seeking food. By coincidence, one ant chooses trail A and the other ant chooses trail B as shown in Figure 4.2, and since trail A is shorter than trail B, the ant that chose trail A reaches the food source first.

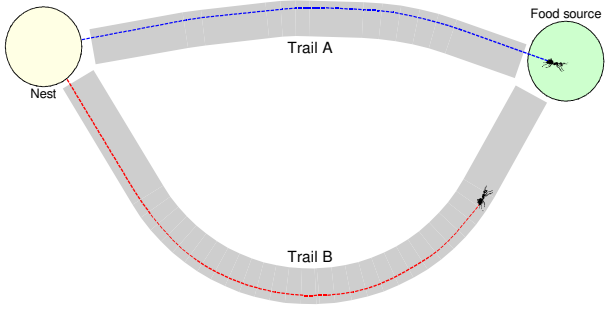


Figure 4.2 One ant reaches the food source.

A moment later — after having gathered enough food — the ant decides to go back home as illustrated in Figure 4.3. However, this time there is *not* a 50-50 chance of choosing one trail over the other. The chance of choosing trail A has changed to *slightly* more than 50%, since the ant itself has marked the trail with its own pheromones. Moments after the "quick" ant has left the food source, the other ant arrives at the food source — this is also shown in Figure 4.3.

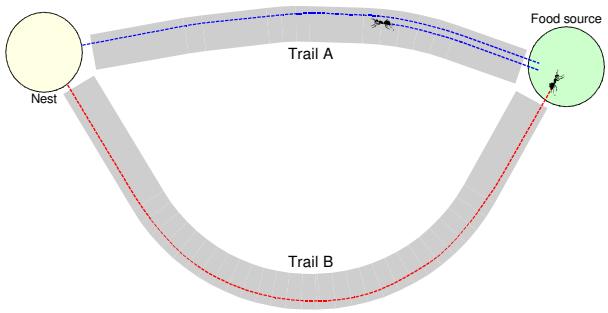


Figure 4.3 The second ant reaches the food source — the first ant is on its way home.

When the "slow" ant decides to make its way back to the nest, the chance of choosing trail A is still a bit higher than the chance of choosing trail B, since trail A has been used twice and trail B only once. So, in this example the slow ant also chooses to use trail A for its way back to the nest (see Figure 4.4).

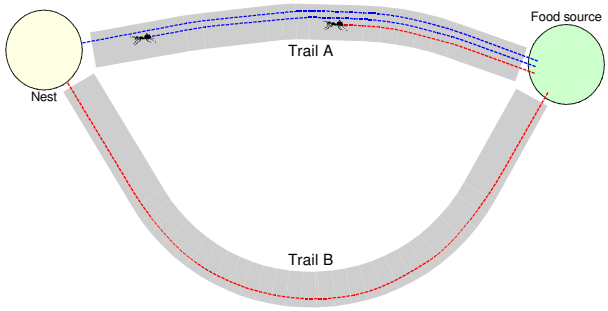


Figure 4.4 Both ants choose the same trail on their way from the food source to their nest.

This will repeat indefinitely, and after a while the chance of choosing the long path (trail B) will become much lower than the chance of choosing the shortest path (trail A) — i.e. the ants have discovered the shortest path between their nest and the food source.

4.3.2 The shortcut problem

In addition to illustrating the fundamental trail-laying mechanism of ants, the example shown in Section 4.3.1 also hints one of the potential problems — the *shortcut problem*. This means that once the ants have found the best route, they are very

reluctant to use alternative routes even though they are only a little bit longer. Furthermore, if a new — and shorter — trail is somehow created, it will take a lot of time before the ants discover this trail. However, this problem can effectively be suppressed by constantly adding a small amount of noise to the system — for instance by making sure that a few of the ants choose their trail completely random, and not by using the strength of the trail's scent (see also Section 8.1.8).

4.3.3 Additional features

Artificial ants can be used for additional purposes such as load balancing and fault recovery. Load balancing can be obtained by urging the artificial ants to stop using trails that are already too busy (see Section 8.1.5), whereas fault management is an intrinsic property of the artificial ants. The speed of the recovery from faulty conditions can, however, be improved greatly using a few simple techniques (see Section 8.6).

4.4 Summary

This chapter has introduced two ideas for implementing distributed and autonomous resource management:

- Software agents
- Simulated ants

Using software agents does not by itself provide any solutions — it should be considered more as a framework (or paradigm) for implementing network control and management functions. More about agents can be found in the Chapters 5, 6, and 7.

The behaviour of ants, on the other hand, suggest a specific solution to problems such as routing and fault management (and, with a few additions, even load balancing). The results of a thorough investigation of the use of artificial ants in telecommunication networks are given in Chapter 8.

Chapter 5

Introduction to Software Agents

This chapter gives a general introduction to software agents and their characteristics and use. In the first part of the chapter, the term "agent" is defined — based on a proposal by M. Wooldridge and N. Jennings [38] — and a list of typical agent characteristics are given. In addition, the different terms used in the agent society are briefly described. Chapter 6 will dig deeper into the details of an agent platform and language as it is defined by FIPA[1].

In the remaining part of the thesis, the terms "agent" and "software agent" will be used interchangeably.

5.1 What is an agent?

Most agent developers have their own opinion on exactly what constitutes an agent [43]. Actually, the question "What is an agent?" is embarrassing for the agent-based computing community in just the same way as the question "What is intelligence?" is embarrassing for the AI (Artificial Intelligence) community [38]. In spite of this lack of precise and concise definitions, this section will try to give a short overview of software agents and their characteristics.

Software agents may — at their simplest — be seen as independent pieces of executing code capable of acting autonomously in an environment with expected and unexpected events. They are often implemented as multiple threads or processes. The main difference between software agents and ordinary programs is the ability of the agents to interact with their environment without direct human interaction at run-time [44].

[1] FIPA: Foundation for Intelligent Physical Agents

5.1.1 Agents and object-oriented programming

Agent programming can be thought of as the next step in the evolution from structured and object-oriented programming [45]. In object-oriented programming the variables and the methods (or functions/procedures) are kept inside appropriate objects, and in addition to operating on arguments the methods operate on the object itself. In this way calling a method in one object *from* another object may be seen as a way of passing information to and/or from the object. This is illustrated in Figure 5.1 where object A sends some numbers to object B by calling b.replace(*some number*) and receives the previous number sent to B as the return value (object B is actually a FIFO with a depth of one).

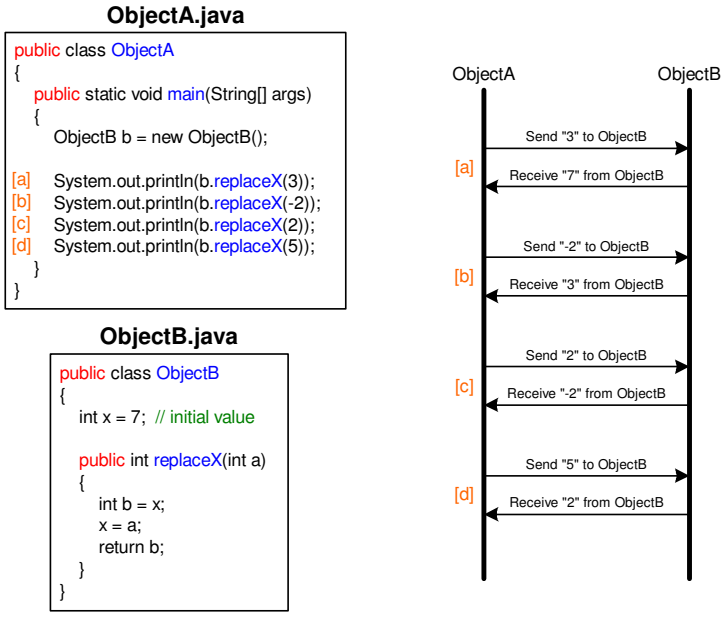


Figure 5.1 Communication between two simple Java objects.

Considering method calls as a way of passing information can be very useful when providing software agents with communication capabilities — this is particularly evident when Java is used as the programming language (see Section 7.2.1).

5.2 Agent characteristics

Software agents are almost always implemented as software objects with some extra properties. These extra properties can be as follows (not all need to be present) [38, 46]:

Autonomy Agents may operate with very limited human intervention or with no human intervention at all. This means that the agent must be provided with rules telling how to cope with any event.

Social ability Agents may interact with other agents or possibly humans using some kind of agent communication language.

Reactivity Agents perceive their environment and respond to changes in timely fashion. The environment may be the physical world, a user (via a graphical user interface), other agents, etc.

Pro-activeness In addition to simply responding to changes in the environment, agents may exhibit goal-oriented behaviour and take the initiative.

A software process exhibiting the listed set of properties is a simple way of defining an agent, and this weak notion is widely used [38]. The *central* concept, however, which distinguishes software agents from normal computer programs is their interaction with their environment [44].

5.2.1 Additional agent characteristics

Several extra characteristics are often used to define agents. Among these are the following [38]:

Mobility Some agents have the capability of stopping execution at any time and transforming itself (and all current state variables) into a bit-stream. In this way it can be sent over a network link and resume execution from where it stopped.

Veracity In order to assure usefulness and that the agents are well-behaved, it is often assumed that they will not (knowingly) send false information to other agents.

Benevolence This is the assumption that an agent does not have conflicting goals, and therefore it will always try to achieve what is asked of it.

Most agent systems do *not* make use of mobility — presumably due to the security issues that will arise from having agents moving around from platform to platform (like worms and viruses).

The decision whether or not to use mobile agents (disregarding the security issues) when designing an agent system, is of course dependent of the nature of the problem that has to be solved. In some cases it is easier to send the agent to the data that needs to be processed, instead of transmitting the data to the agent.

An example where mobile agents could be useful, could be when searching huge databases for information (articles, books, etc.). Instead of sending the entire con-

tents of the databases over the Internet before performing the search operation, it would be more practical to send agents to the location of the databases, perform a search locally, and let the agents (containing the results) return to the user. Of course, existing literature databases will never send the entire contents of their databases to a non-trusted user. Instead the user sends a list of keywords to the database, and a list of matching documents is sent back to the user. The use of a mobile agents, however, adds a lot of extra possibilities, since an agent can contain executable code implementing its own search methods instead of relying entirely on the search methods provided by a literature database.

Most Internet search engines are greatly dependent on so-called "robots" (also called "spiders" or "crawlers") to collect information from web servers [47]. However, in spite of their misleading name, search "robots" do *not* move from place to place and have therefore nothing in common with mobile agents. They are merely static programs downloading whatever documents they can find on the Internet (HTML, PDF, Microsoft Word documents etc.). These found documents are then indexed and placed in a database by category and keywords. I.e. this is an example where the documents are sent to the processing location contrary to article databases and electronic libraries.

5.3 Agent toolkits

Certain basic functions are very useful in any agent system independent of the type of agent system. These functions are:

- Agent communication language
- Yellow pages service
- Agent management
- Mobility

An agent toolkit[2] provides methods to do all these things thereby greatly reducing the workload of the agent developer. If no agent toolkit is used, all these mechanisms must be implemented by the developer.

[2]The terms "agent toolkit" and "agent platform" mean exactly the same thing throughout this thesis.

5.3.1 Agent communication language

A common agent communication language (ACL) enables the communication among heterogeneous agents. Agent communication theory has tended to be based on speech act theory [38]. The sender may be compared to a speaker trying to influence the listener (the receiver) through the act of speech and make him react in the way the speaker intends to. Just like the speech acts may fail in the physical world (the listener may decide not to react in the way that the speaker intends) they may also fail in the agent world. Hence, agent communication must be capable of dealing with things like mis-understandings. In speech act theory the term *performative* is used to identify the intention behind spoken communication [44]. Examples of performatives could be verbs such as *request*, *reply*, *inform*, or *refuse*.

The most popular agent communication languages [46] are Knowledge Query and Manipulation Language (KQML) [48] and FIPA ACL. In recent years, however, FIPA ACL has become the primary choice since FIPA ACL is only a part of the whole FIPA framework which includes mechanisms for virtually every aspect of software agent design — perhaps most importantly, FIPA includes an agent management system [49].

KQML, on the other hand, was among the first initiatives to specify a speech-based interaction of agents [44, 45]. Even though KQML has been widely deployed in many projects it has a number of drawbacks. First of all there is no consensus in the agent community on a set of specifications. In addition KQML lacks well-defined semantics, which means that different KQML-based agent systems will normally not be able to communicate directly because they speak different dialects [44].

In addition to an agent communication language, an agent toolkit also needs to provide a mechanism for encoding and transporting the messages. FIPA's approach to this will be described more in detailed in Section 6.4.

5.3.2 Yellow pages services

In an agent system consisting of a variety of agent types, it is more or less necessary to appoint at least one special agent with the task of taking care of yellow pages services. This means, that if an agent wants another agent to carry out a specific task, it can first look up (using the yellow pages service) the names of the agents that are indeed capable of carrying out that task. Also, when an agent starts its life cycle, it must register the services it offers.

5.3.3 Agent management

When a software agent is "brought to life" one of the first things it needs to do is to register itself at the agent management system to make its existence known to other agents. The agent management system is responsible for the white pages services — i.e. it maintains a list of the names of all agents together with their location.

An agent management system may also take care of other tasks like, for instance, life cycle management (e.g. the creation and destruction of agents).

Mobility

An agent management system may also provide methods to support various forms of agent mobility. This could be migration of agents, cloning of agents, and invocation of agents at remote facilities.

5.3.4 Commercially available agent toolkits

For agent systems to become successful and widely deployed, agent toolkits need to be widely available and must be robust enough to cope with situations where uptime is critical — e.g. flight control, traffic warning systems, telecommunication management systems, E-business, E-commerce, etc. A number of agent toolkits have become available over the last few years — a Table 5.1 lists a few of them. A much more extensive list can be found in [44].

Table 5.1 Publicly available agent toolkits.

Toolkit Name	Company	Language	Standard	Licence
Aglets [50]	IBM, Japan	Java	MASIF	Open source
FIPA-OS [51]	Nortel Networks	Java	FIPA	Open source
JAFMAS [52]	Uni. Cincinnati	Java	KQML	Unrestricted license
JATLite [53]	Stanford Uni.	Java	KQML	GNU public license (GPL)
Open Agent Architecture [54]	SRI	ANSI C/C++, VB, Lisp, Prolog, Java, Perl, Delphi	KQML	Academic

It is quite apparent, that Java is widely used in the agent society. This is for a good reason, since Java has proven itself to be genuinely platform independent and it supports many features that enable fast software development (good error reporting at run-time, built-in garbage collector, strict types, etc.).

5.4 Summary

In this chapter software agents and their characteristics have been introduced. Agent development may be seen as the next step in the evolutionary process from structured programming over object-oriented programming. However, there is more to agents than simply software object with a few extra properties. When creating an agent system, the agent developer will normally start by selecting an agent toolkit, that provides the necessary infrastructures needed for agent communication, agent management, mobility, etc.

Chapter 6

A standardized agent framework

This chapter deals with agents as they are defined by FIPA [55] (Foundation for Intelligent Physical Agents). FIPA was formed in 1996 as a non-profit organization in an attempt to produce a set of standards covering most aspects of agents. The official goal of FIPA is to *promote technologies and interoperability specifications facilitating end-to-end interworking of intelligent agent systems in modern commercial and industrial settings* [56].

Several versions of the FIPA specifications exist. This chapter contains an extract of the more than 40 FIPA specifications published in year 2000.

6.1 The agent reference model

The FIPA 2000 agent reference model is shown in Figure 6.1 [49]. In this figure, "software" covers any non-agent executable accessible through an agent. Agents may access software for adding new services, learning new communication protocols, gaining information from the environment etc. Also, the entities in the figure describes *logical* entities — i.e. they do not imply anything about the physical configuration. This means that the agent platform developer is free to combine some of the functions into one agent.

An agent is (naturally) a fundamental actor on an agent platform (AP). All agents must have a unique agent identifier (AID) to make them distinguishable unambiguously in the agent universe. Also, all agents must have at least one owner (organization, human name, etc.) and it may have certain capabilities for accessing external software. The message transport service (MTS) is a service provided by an entity called the agent communication channel (ACC) [57]. The MTS provides the default communication method between agents on different platforms

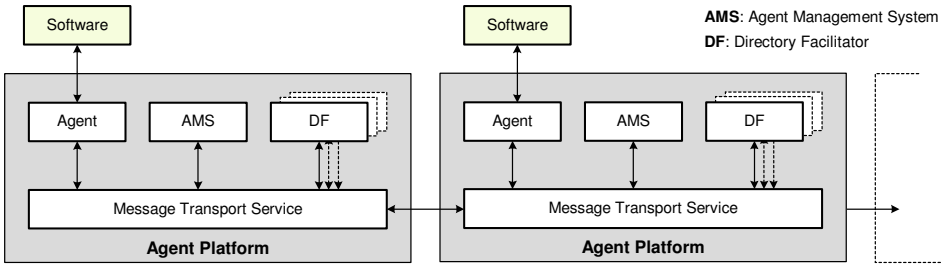


Figure 6.1 FIPA agent reference model.

(see Section 6.5).

Yellow pages services are provided by the so-called directory facilitator agent, or simple the DF. The DF is a mandatory component of an agent platform, and multiple DFs (perhaps cooperating) may exist within an AP. Agents may register at the DF to make their capabilities known to other agents.

The agent management system (AMS) is also mandatory. Only one AMS may exist within a single agent platform. The AMS performs supervisory functions — i.e. it controls the access to and the use of the agent platform. All agents must register at the AMS to obtain a valid AID, and the AMS offers white pages services to other agents.

The agent platform (AP) provides the physical infrastructure needed by the agents. It consists of the computer, the operating system (UNIX, Microsoft Windows, etc.) the mandatory components (DF, AMS, and MTS) and a number of agents. An agent platform does not necessarily have to be resident on a single computer; an agent platform may be distributed over several processors or computers using proprietary or standard middleware protocols like, for instance, CORBA. Even though FIPA provides standards for how agents may communicate with each other they are free to exchange messages directly using any method supported by them. However, the use of proprietary mechanisms for communication goes against the idea of trying to set up these standards.

6.2 Agent management services

6.2.1 Agent naming

An agent must be unambiguously identifiable through its AID. The AID consists of a pair of parameters. One parameter contains the name of the agent and the other parameter contains additional information. The additional information can

be the address of the agent *or* alternatively an AID of a resolver. An example of an AID is shown in the following[1]:

```
(agent-identifier
  :name AgentA@nonameA.com
  :resolvers (sequence
    (agent-identifier
      :name ams@nonameB.com
      :addresses (sequence iiop://nonameB.com/acc))))
```

The name of the agent (`AgentA@nonameA.com`) does not necessarily say anything about where this agent can be found — if it is a mobile agent, it may move from AP to AP. Normally the text following the "`@`" character is the location of the agent's home agent platform (HAP), however the important thing is that the name must be unique. The AID in this example does not contain an address of AgentA. Instead it contains a list of so-called resolvers — i.e. agents knowing where AgentA can actually be found. The list of resolvers in this case only includes a single agent which is the AMS (offering white pages services) named `ams@nonameB.com`. The resolving agent can be contacted at the address `iiop://nonameB.com/acc`. The prefix "`iiop`" refers to the Internet Inter-ORB Protocol which is a part of the CORBA specification [59].

6.2.2 The directory facilitator

The directory facilitator (DF) is a mandatory component af an agent platform. The DF is a trusted agent offering the registration of services provided by other agents. "Trusted" means that the DF must do whatever it can to make sure that the information it holds about other agents is as accurate and timely as possible. Furthermore, the DF must offer equal access to its directory to all authorized agents. Each agent platform must have at least one DF, however, any number of DFs (larger than zero) may exist and DFs may share information with each other and form federations.

Each DF must be able to perform the following functions:

[1] The notation used here corresponds to the string-based FIPA ACL message representation [58]

register Every agent wishing to make its services known to other agents may do so using the *register* function of the DF. Registering services at the DF does not imply any future commitment or obligation to carry out an announced service — an agent *can* refuse a request for announced services. Also, the DF can not guarantee the correctness of the information being registered. It has to rely on the information given by other agents.

deregister If an agent does not offer a service anymore (or if the agent does not wish to make its services publicly known) it may use the *deregister* function to remove the directory entry from the DF.

modify At any time an agent may modify an announced service using the DF's *modify* function.

search Authorized agents may use the DF's *search* function to look up information in the directory. Once again, the DF can not give any guarantees about the validity of the response to the search request, since it relies on information given by other agents.

6.2.3 The AMS agent

The FIPA agent platform must have exactly one agent management system (AMS). If the agent platform spans over several machines then the AMS still represents the authority across all machines. The AMS is responsible for managing a lot of things regarding the operation of the agent platform. For instance it may create agents[2], delete agents, and — if the platform supports mobility — supervise the migration of agents to and from other platforms.

In order for agents to get access to an AP and the message transport service (MTS) of the AP it must register with the AMS of the agent's HAP and give a description of itself, whereby it obtains a valid AID (if the agent is granted access to the AP).

As for the DF, the AMS supports *register*, *deregister*, *modify*, and *search* functions. Using these functions an agent can, for instance, modify its description at any time or it can search for other agent descriptions. Additionally, the AMS can instruct its AP to perform a number of operations on the agents residing on the platform. Examples of these operations are: suspend an agent, resume agent execution, and terminate an agent.

[2]The AMS is not the only agent capable of creating agents.

6.3 The agent platform

The agent platform does not actively do anything by itself. It merely offers a location where agents can reside and an infrastructure that can be used by the agents.

6.3.1 Agent life cycle

The life cycle of an agent is managed by the AP. Figure 6.2 shows a state diagram with all the possible states of a FIPA compliant agent.

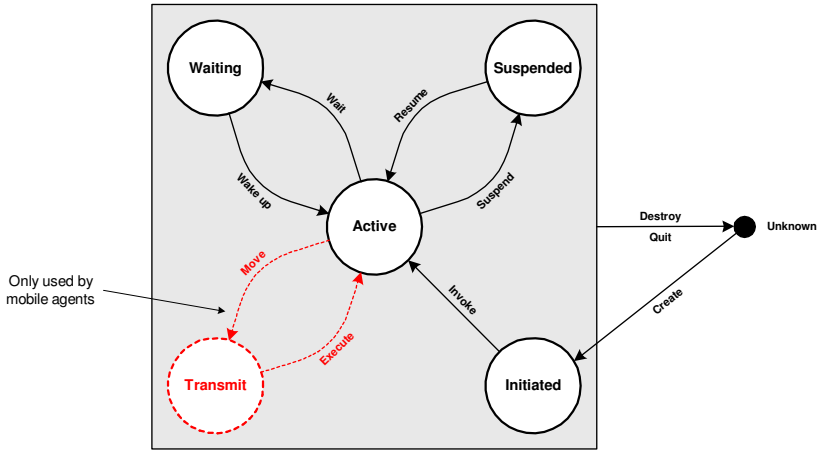


Figure 6.2 The life cycle of a FIPA compliant agent.

When an agent is in the `active` state the MTS delivers messages to the agent as normal, but when an agent is in any of the states `initiated`, `waiting`, or `suspended` the MTS buffers messages sent to the agent until it returns to the `active` state. Alternatively, the MTS may forward messages to another location if a forward has been set up for the agent.

In order to support mobile agents, the FIPA agent may also enter a `transit` state. Agents that are in transit can not receive messages, so the MTS buffers messages until the agent re-enters the `active` state — either on another platform, if the transit was successful, or on the same platform, if the operation failed. If the agent has moved to another platform, its messages will be forwarded by the MTS to the new location.

Some transitions between states can only be initiated by the agent itself, whereas other transitions can only be initiated by the AMS. An agent can be terminated

gracefully through its *quit* function. However, if an agent ignores *quit* it can be terminated by force (through `destroy`) by the AMS.

6.4 FIPA agent communication

Due to the lack of agent communication language standards FIPA developed its own ACL as an attempt to create a universal message-oriented communication language. The main purpose of FIPA ACL is to provide a way to package messages so that the meaning of a message is clear to other FIPA compliant agents. This section gives an overview of the overall structure of FIPA ACL messages.

6.4.1 FIPA ACL message structure

FIPA ACL messages contain one or more message elements. Whether or not an element is needed depends on the situation — only the *performative* is mandatory in all ACL messages. Table 6.1 shows the full set of FIPA ACL message elements together with some simple examples [60].

Table 6.1 FIPA ACL message elements.

Message element	Category of elements	Example
`performative`	Type of communicative acts	`propose`
`sender`	Participant in communication	`AgentA@noname.com`
`receiver`	Participant in communication	`AgentB@noname2.com`
`reply-to`	Participant in communication	`AgentC@noname.com`
`content`	Content of message	`(= (iota ?x (p ?x)) a)`
`language`	Description of content	`FIPA SL0`
`encoding`	Description of content	—
`ontology`	Description of content	`brokerage-agent`
`protocol`	Control of conversation	`fipa-request` (see Section 6.4.3)
`conversation-id`	Control of conversation	`vod-brokering-1`
`reply-with`	Control of conversation	`query1`
`in-reply-to`	Control of conversation	`query1`
`reply-by`	Control of conversation	`15:28:00`

The performative describes the *intention* of the conversation. The English language contains several hundred performatives [44], but FIPA has cut that number down to a minimal set of only 22 performatives used in the world of agent communication [61] (see Appendix B).

The sender and receiver are self-explanatory, whereas `reply-to` can be used to instruct an agent to reply to an agent *other* than the sender of the message.

The language, encoding, and ontology[3] *describe* the content of the message (the content itself is contained in the `content` element). FIPA specifies several content languages: FIPA Semantic Language [62], FIPA Constraint Choice Language [63], FIPA KIF Content Language [64], and FIPA RDF Content Language [65].

KIF, Knowledge Interchange Format, is an American National Standard intended for interchanging knowledge among widely different computer systems. Similarly, RDF — Resource Description Framework — is a W3C standard providing a lightweight system to support exchange of knowledge over the Internet using XML.

6.4.2 FIPA ACL representations

Before a message can be sent from one agent to another (normally through the MTS), it must be encoded into a format readable by the receiver. FIPA defines three ways of encoding messages:

- Binary representation [66]
- String representation [58]
- XML representation [67]

The string representation is particularly well suited for situations where agent messages should be easily readable by humans. The binary representation, on the other hand, is a much more compact encoding scheme, and therefore well suited for saving bandwidth. XML is a format widely supported by almost any programming language, so parsing messages in XML format is extremely easy.

6.4.3 Interaction protocols

Typically, the sequence of messages in a conversation between agents fall into a number of patterns that are repeated over and over again. At any point in a conversation an agent expects certain messages (or one specific message) to follow the previously sent (or received) message. These typical patterns of agent messages are called *interaction protocols* [68].

[3] An ontology defines concepts and relationships in a given domain. Agents sharing a common language and an ontology can communicate effectively both syntactically and semantically [44].

A number[4] of standardized FIPA-compliant interaction protocols (IPs) have been defined to avoid mis-understandings or ambiguities. The IPs cover topics like requests, queries, auctions (English and Dutch), brokering, etc. Three of these FIPA-compliant interaction protocols are shown in Figure 6.3.

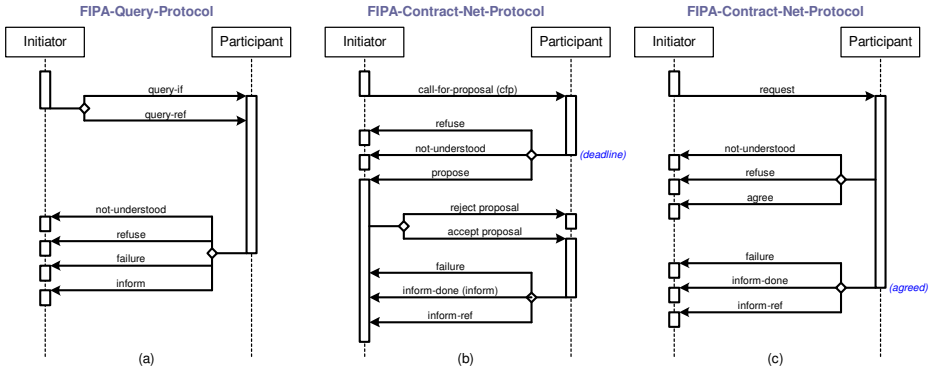


Figure 6.3 Examples of standardized FIPA interaction protocols. (a) FIPA Contract Net Interaction Protocol; (b) FIPA Query Interaction Protocol; (c) FIPA Request Interaction Protocol.

An agent does *not* need to implement any of the standard IPs to be FIPA ACL-compliant — the standard IPs are merely meant as a help to the agent developer. Moreover, agents are not restricted from using implementation specific IPs with others names, however, if a standard name is used for an IP, the agent must behave according to the FIPA IP specifications.

6.5 Agent message transport

FIPA agent messages can be transported using the message transport system (MTS) provided by an entity called the agent communication channel (ACC) [57]. The ACC may offer some conveniences like buffering of messages, error handling, etc.

Before messages are sent, they are placed in a *message envelope*. The message envelope contains a collection of parameters as well as the payload which is the actual agent message. Mandatory parameters in the message envelope are:

[4] At the time of writing FIPA has defined 10 interaction protocols.

`to` The intended receiver of the message.

`from` The sender of the message. By setting this parameter, the sender may receive confirmation messages (or error message) from the ACC.

`date` The date.

`acl-representation` This information, intended for the final recipient of the message, tells how the ACL message in the payload is encoded.

Additional parameters can be used to support encryption of messages, tracking the trail of the message, etc.

FIPA supports three ways agents can send messages to other agent on a remote agent platform (see Figure 6.4) [57]:

1. Agent A sends a message intended to agent B to its local ACC. The ACC then takes care of sending the message to the correct remote ACC, which, in turn, forwards the message to agent B.

2. Agent A sends a message (intended for agent B) *directly* to the ACC located on the same platform as agent B. Again, the remote ACC forwards the message to agent B. This communication method can only be used if agent A supports one of the interfaces of the ACC on the remote platform.

3. Agent A sends a message *directly* to agent B bypassing the ACC on both the local and the remote platform. Message buffering, error handling, etc. must be handled by the agents themselves.

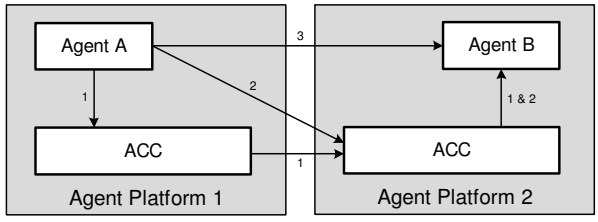


Figure 6.4 The three methods of inter-platform agent communication supported by the FIPA framework.

6.5.1 Transport protocols

A message transport protocol (MTP) is used to carry out the actual transfer of messages between two ACCs (on different platforms). FIPA has defined a set of standards describing message transport protocols using the following network protocol:

- Internet inter-ORB protocol (IIOP) [69]
- Wireless application protocol (WAP) [70]
- Hypertext transfer protocol (must be HTTP/1.1) [71]

6.6 Summary

This chapter has given a short introduction to the agent framework described by FIPA. FIPA describes basic things like how agents can communicate, the contents of agent communication messages, yellow- and white-pages services, agent management, a set of standardized message sequences (interaction protocols), etc. In many cases, the FIPA specifications do not *have* to be followed, so proprietary solutions may be used side-by-side with FIPA-compliant solutions.

The main purpose of FIPA is to make it easier to build agent-systems that are interoperable — also between different vendors. FIPA does *not* interfere with the internal design of software agents. This is entirely the responsibility of the agent developer.

Chapter 7

The IMPACT Project

The IMPACT project[1] was a two year project sponsored by the European Commission under the ACTS (Advanced Communications Technology and Services) project AC324. The project was green-flagged in February 2000 at a live demonstration that took place at Tele Danmark in Aarhus, Denmark. The partners in the project were:

- Queen Mary and Westfield College, United Kingdom
- Swisscom, Switzerland
- Flextel, Italy
- National Technical University of Athens, Greece
- Teltec, Ireland
- Tele Danmark, Denmark

This chapter describes some aspects of the IMPACT project and observations done during the two year period [72–79].

7.1 Objectives

The main objectives of the IMPACT project was to develop a framework for a society of software agents capable of carrying out different control and management plane functions in an ATM based network. The original idea was to use agents

[1] IMPACT: **Imp**lementation of **a**gents for **C**AC on an ATM **t**estbed

to control the capacity of virtual paths in a network, and to demonstrate scenarios where the agents would be capable of utilizing the resource in a more efficient way by re-arranging the resources or even by re-routing connections while they are in use. A simple example of a network controlled by the IMPACT system could be the one shown in Figure 7.1, where users A and B may choose between two already established virtual paths when setting up a connection to the users C and D. There is only one virtual path between the edge switches and the users, since most ATM network interface cards (NICs) only support one virtual path (VPI = 0) which, in turn, is typically capable of carrying 1024 virtual channels.

As shown in Figure 7.1, the core switches in the IMPACT system never switch virtual connections — this is only done in the edge switches. Moreover, loopback connections and local connections (like the connection between user A and user B) are not routed through the VPs managed by the IMPACT system. However, the IMPACT system is fully capable of setting up local and loopback connections also.

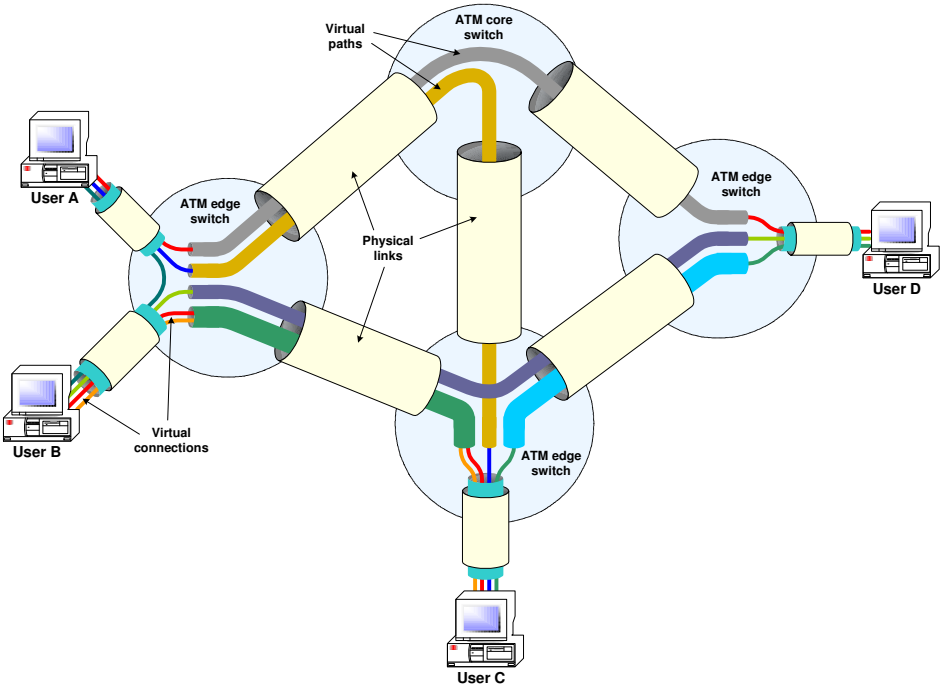


Figure 7.1 Four terminals connected to an ATM network with five virtual paths set up in the core network and six virtual connections.

In addition, when the ideas for the project were conceived in 1997, many people believed that ATM had a good chance of replacing the Ethernet and Token Ring

LAN technologies in the long run. This — in addition to ATM's excellent properties in backbones networks — was the main reason for choosing ATM as the protocol to use in the project. However, even though it now seems like ATM will never become widely available in LAN environments, the concepts in the IMPACT project might just as well be used with other connection-oriented protocols, like — for instance — MPLS.

7.2 The IMPACT agent platform

One of the first tasks of the IMPACT project was choosing an appropriate agent toolkit. The requirements were [46]:

- FIPA compliance
- Support for multiple threads
- Flexible and modular design
- Access to source code
- Support and detailed documentation

A review of the available platforms (JATlite, JAFMAS, and Grasshopper) was made and this led to the following conclusions [74]:

- No FIPA-compliant platforms were available (at that time)
- Some solutions were overly complex with too much overhead
- Agent communication mechanisms were poorly defined

Of course, at lot has changed since then. Many platforms have become available, and if the IMPACT project was to start all over again, the FIPA-OS (see Section 5.3.4) would probably have been used, since it has all the desired features.

Since there was not enough time to wait for an appropriate platform to become available, it was decided early on that a proprietary platform had to be developed. This platform — the Java management agent platform (JAM) — was developed by Swisscom who has also used it for projects within Swisscom [46]. The basics of JAM will be described in this section.

7.2.1 JAM agent platform basics

The JAM agent platform is fully implemented in Java. All JAM agents inherits from a base class called JAMAgent, which provides communication basics and a generic graphical user interface. Additionally, each agent inherits from the class JAMWorker which, in turn, inherits from the Thread class (a standard component in Java).

The high-level architecture of a generic JAM agent is shown in Figure 7.2. All incoming FIPA messages are placed in a request queue and the agent returns immediately. Each element in the request queue is handled by a separate (light-weight) thread and after having processed the request the agent may send a FIPA message to another agent and/or to the agent where the request came from.

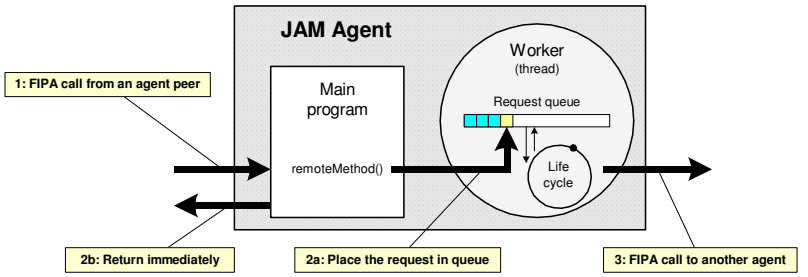


Figure 7.2 Architecture of a generic JAM agent.

7.2.2 FIPA ACL and the JAM platform

The supported number of FIPA message parameters and performatives in the JAM platform was cut down to a minimum to accommodate the specific needs of the agents in the IMPACT system. Table 7.1 lists the message parameters implemented in JAM as well as their primary use. Other parameters like, for instance, the `protocol` and the `language` parameters *should* have been included too, in order to obtain full FIPA compliance. However, since communication between IMPACT agents and non-IMPACT agents was not envisaged — and given IMPACT's time constraints — the values of those parameters was implicitly assumed [46].

The 22 FIPA performatives (see Appendix B) were cut down to only 8 performatives. These 8 performatives and their main functions are shown in Table 7.2.

JAM makes extensive use of the features provided by the Java language. These are Java RMI (Remote Method Invocation) and Java's Reflection API. The Reflection

Table 7.1 FIPA parameters implemented in the JAM framework.

Message parameter	Use
`performative`	Identify the intention
`sender`	Identify the sender
`receiver`	Identify the receiver
`content`	Content of the message

Table 7.2 FIPA performatives in the JAM framework.

FIPA message performative	Main function
`call for proposal (cfp)`	Negotiation
`cancel`	Action performing
`failure`	Error handling
`confirm`	Information passing
`inform`	Information passing
`reject proposal`	Negotiation
`accept proposal`	Negotiation
`propose`	Negotiation

API is actually not recommended for use in "normal" applications — is it mainly intended for writing Java debugger tools, class browsers, or GUI builders. The reason that it should be avoided in normal applications is that the speed of Java Reflection is (or better, *was*) very poor. Luckily, the speed of Java Reflection has improved greatly — actually the performance has been improved by a factor of 20 from Java version 1.3 to Java version 1.4 [80]. In any case, Java Reflection turned out to be very useful in the JAM toolkit.

Java RMI

In the JAM framework FIPA messages are sent using Java RMI. Each RMI server object defines an interface which can be used to access the object from other computers. In order for the client to find the methods in the server object, Java RMI depends on a mechanism called *RMIRegistry* which is a process running on the server containing information about the interfaces of server objects.

This is illustrated in Figure 7.3. HostB runs an object (ObjectB) containing three methods accessible through RMI. ObjectA running on Host A can call these methods just as if ObjectB was running on the local computer. If methods in ObjectA are to be accessible from ObjectB, the RMIRegistry process has to run on Host A as well.

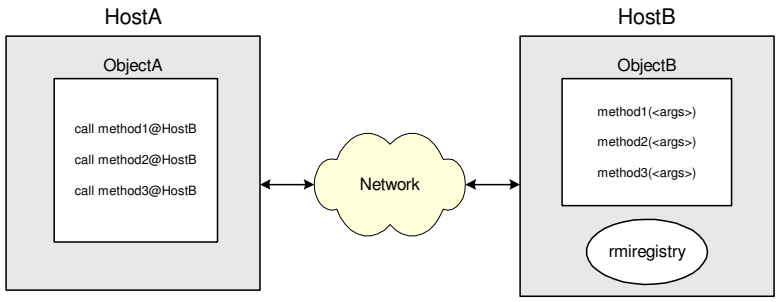


Figure 7.3 With Java RMI methods in remote objects are called just like methods in local objects.

The arguments used when calling remote methods may be simple data types (integers, floats, strings, etc.), but actually complete objects (even objects containing executable code) may be passed over a network through RMI. This is accomplished through Java's "Serializable" interface. When creating a Java object implementing the "Serializable" interface, the entire object can be transformed into a bit-stream suitable for transmission from one computer to another.

Java RMI does some of the same things as CORBA does. CORBA, however, is much more flexible since it provides things like language independence, load balancing, and fault tolerance. A comparison of CORBA and Java RMI (and DCOM) is given in [81].

Java Reflection

In addition to Java RMI the JAM agent platform uses some features of Java's Reflection API for agent communication. As mentioned earlier, all agents in the JAM framework inherit from the JAMAgent. The JAMAgent implements methods to handle all of the 8 performatives listed in Table 7.2 and in this way all JAM agents can understand messages with any of these performatives.

However, this is not adequate to guarantee, that an agent actually understands the *contents* of a message. In order to determine whether or not an agent understands the contents of an ACL message, the JAMAgent also implements a *signature matching* method. The contents of JAM messages are Java objects, and upon reception of a message the signature matching method uses two methods from the Java Reflection API (`getDeclaredMethods()` and `getParameterTypes()`) to check whether or not the specific agent contains any methods capable of handling this type of object. If there is a match, the matching method is called — otherwise the message is not processed by the agent.

This approach adds a lot of dynamics to the agent system. Normally in programming, method names have to be known at compile time in order to be able to use them. By using Java's Reflection API it is possible for agents to communicate without knowing actual remote method names and instead rely on the set of FIPA performatives.

7.2.3 White pages service in JAM

In JAM the white pages service is implemented by the so-called *directory facilitator*. This is in contrast to the FIPA standards, where the white pages service is performed by the agent management system whereas the FIPA directory facilitator takes care of *yellow* pages service. JAM, however, does not provide any yellow pages service — instead the function of each agent is implicitly given by the type and the name of the agent.

7.2.4 The JAM graphical user interface

The JAM agent toolkit provides a generic GUI to all agents. Figure 7.4 shows an screenshot of a specific agent (the proxy user agent, PUA) in the IMPACT system.

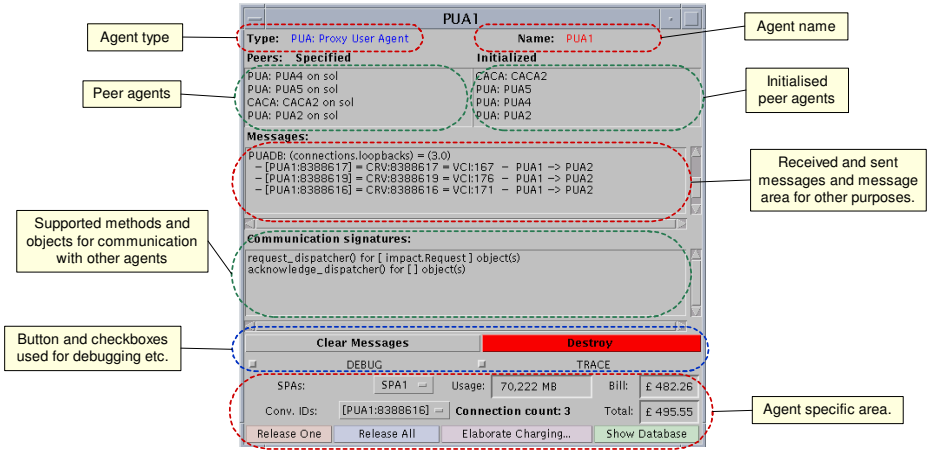


Figure 7.4 The JAM graphical user interface.

The upper part of the GUI displays the agent's name and type as well as all the known peer agents and information about whether or not the peer agents are actually alive (initialised) or not. Furthermore there is a scrollable text area where agent messages and other relevant logging information are shown. Different text colours are used to indicate different error conditions (RMI problems, initialisation

problems, etc.). Finally, the generic part of the JAM agent GUI has a button to stop the agent and two checkboxes useful for debugging or for tracing the interaction between agents.

The lower part of the GUI may be used freely by the agents to display information specific for the agent type or to add extra controls that can be used for special purposes. For instance, the agent shown in Figure 7.4 shows things like billing information, used Q.2931 conversation IDs, number of ATM connections, etc. In addition, it is possible to release one (or all) connection(s) directly from the PUA if desired, and an extra window can be opened, where the rationale behind the charging and the selection of service provider for a specific connection is elaborated.

7.3 The actors in the IMPACT system

The necessary tasks have been split out between the agents much like the situation in a competitive, deregulated telecommunications market, with a network provider (or several network providers) and competing service providers. The different agent types in the IMPACT system will be described briefly in the following part of this section. Some of the functions of a few selected agents will be described in more detail later on in this chapter.

7.3.1 Directory Facilitator

As mentioned in Section 7.2.3 the agent system has one special agent called the *directory facilitator* which offers white pages services. I.e. when an agent is invoked it must register at the directory facilitator to make its presence known to the other agents in the system.

The directory facilitator is a critical part of the system, since without it the agents will not be able to communicate. It would, however, be relatively easy to extend the JAM agent platform to support several directory facilitators so that this single-point-of-failure problem would be eliminated. Implementation of backup directory facilitators was considered as low priority, due to the fact that the IMPACT system is only a prototype, and due to time constraints.

7.3.2 NPA — Network Provider Agent

The network provider agent (NPA) is the owner of the physical equipment — i.e. ATM switches, cables, fibres, etc. The NPA is the only agent with the full knowledge about the network topology at the physical level. In this framework the job of the network provider is to create (and destroy) virtual path connections (VPCs)

using the switch wrapper agents, and lease these VPCs to the service provider agents.

7.3.3 SPA — Service Provider Agent

The service provider agents (SPAs) represent the service providers (SPs) in the agent society. The SPAs lease VPs from the NPA, however the responsibility of controlling the resources of the VPs is left to the resource agents (RAs) owned by the SPAs. In addition, the SPAs may bid for connections (requested by the users through the PUAs) even though the user does not have a contract with the service provider. This means, that users might end up using a service provider completely unknown to them.

In the IMPACT project two kinds of service providers were defined. One type is owned by the network provider, the other is not. A service provider owned by the network provider is called NSP (Network Service Provider) in the following, whereas a service provider *not* owned by the network provider is referred to af the SSP (Secondary Service Provider). However, in this chapter this differentiation will not be made.

7.3.4 RA — Resource Agent

The resource agents (RAs) are owned by SPs. Each RA is responsible for the VPs between one source-destination pair — i.e. one RA may control several VPs possible using different paths between the two end-points. If a service provider wants to be able to set up virtual channel connections (VCCs) between n sites the required number of RAs is $\frac{n(n-1)}{2}$, which leads to a bit of a scalability problem (see Section 7.9.1).

RAs owned by the same SP are capable of borrowing resources from each other. For instance, if a VPC controlled by RA1 shares one part of its path with a VPC controlled by RA2 and the other part of its path with a VPC controlled by RA3, RA1 can borrow spare capacity from RA2 and RA3.

7.3.5 CA — Connection Agent

In essence, the connection agent is a mediator between the user (represented by the PUA) and the service providers (represented by the SPAs and their RAs). The CA receives connection requests (and additional instructions) from the PUA. In addition it receives bids for the connection from the SPAs (through the RAs) and selects which service provider to use for the connection.

7.3.6 PUA — Proxy User Agent

The proxy user agent represents the user in the agent world. The PUA communicates with the agents using the FIPA-flavoured ACL. The communication with the user takes place through ATM UNI signalling (UNI-3.1 or Q.2931) [28, 82].

Minor changes were made to the contents of the ATM UNI signalling messages in order to make possible for the user to transmit these extra preferences to the PUA (see Section 7.4.1):

1. Allow re-routing
2. Allow slow connection setup

If a user allows a connection to be re-routed he might occasionally experience connection-loss for a few seconds, while the connection is taken down and re-established through another path by resource agents.

Similarly, if the user is prepared to wait some extra seconds for a connection to be set up (*slow connection setup*) the risk of blocking is reduced, since the resource agents get some extra time to shuffle around resources (or re-route other connections) before setting up this connection.

The PUA does more than simply translating between ATM signalling and FIPA ACL. For instance, it includes mechanisms to help assist the user in choosing the cheapest service provider if more than one bid is received for a connection.

7.3.7 SwWA — Switch Wrapper Agent

The switch wrapper agents are owned by the network provider. Their main purpose is to provide ATM switches from different vendors with a uniform interface easily accessible through FIPA ACL. The SwWAs may be accessed by the NPA (to set up VPC) or by RAs (to set up VCC). In addition the SwWAs may be accessed by RAs to adjust the bandwidth reservations (or UPC parameters) of already established connections. The access to the ATM switches is done through the API (if possible) and through SNMP if the switch API is inaccessible.

7.3.8 PCA — Proxy Connection Agent

Unfortunately, the IMPACT system is not capable of scaling indefinitely. For instance, the number of RAs (owned by one SP) scales proportionally to n^2, where n is the number of sites. In order to improve scalability, the network could be

divided into several domains controlled each controlled by its own society of IMPACT agents.

In order to support inter-domain connections a proxy connection agent (PCA) has been envisaged. However, due to time constraints (and limited testbed facilities) this agent was never implemented.

7.3.9 Relationship between the agents

Figure 7.5 shows the relationship between the agents in the IMPACT system. The PCA is not included here since it was only a *conceptual* agent that was not implemented. The notation used in this figure is borrowed from relational database diagrams.

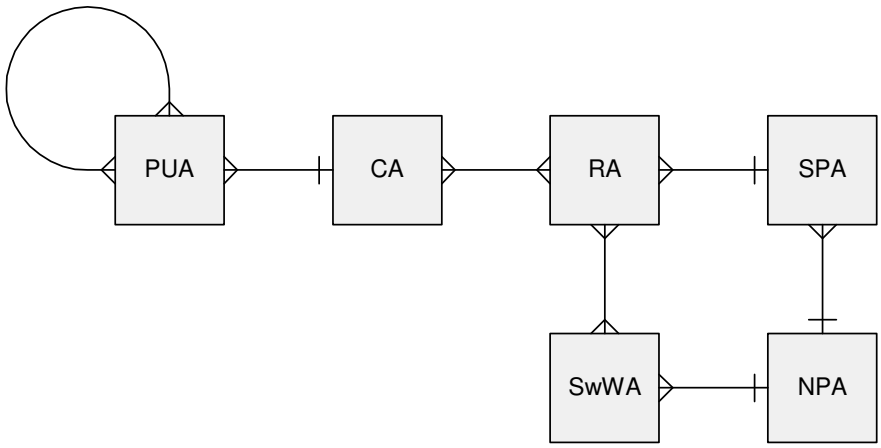


Figure 7.5 Relationship between the IMPACT agents.

All PUAs know of the existence of all other PUAs in the network domain, whereas each PUA only knows of the existence of exactly one CA. The CA may serve several PUAs and it knows of the existence of several RAs. The RAs can communicate with several CAs, but each RA is owned by exactly one SPA. In addition an RA may communicate with several SwWAs (in order to establish VCCs).

The SPAs normally own several RAs, but each SPA can only communicate with one NPA. The NPA may communicate the several SPAs and with several SwWAs (in order to set up VPCs). Each SwWA may be controlled by several RAs, but only by one NPA. I.e. in this model, network providers can not share physical equipment.

7.3.10 Ownership of the agents

The ownership of the agents is illustrated in Figure 7.6.

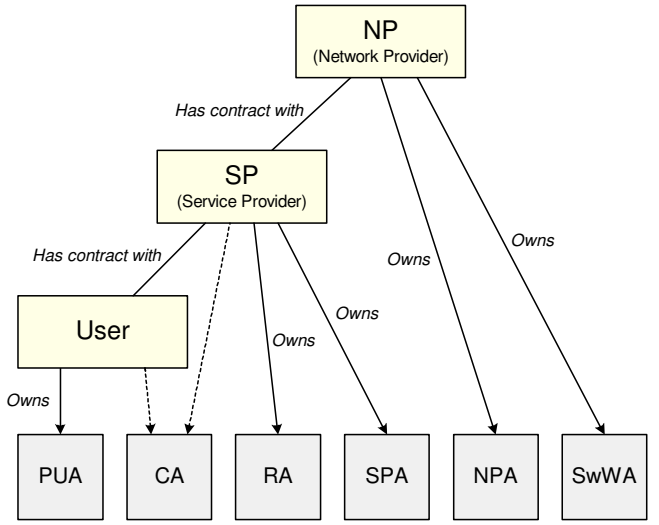


Figure 7.6 Ownership of the agents.

The users have contracts with service providers, and the service providers have contracts with network providers. The NPAs and the SwWAs are owned by network providers whereas the SPAs and the RAs are owned by service providers. Finally, the PUA is owned by the users.

The ownership of the CA is not so clear. The function of the CA is heavily controlled by instructions from the PUA, however one CA serves many PUAs.

7.3.11 Location of the agents

Since the agents use Java RMI for communication most of them could actually be located anywhere, as long as they can be reached through a network. In the IMPACT demonstration system, most agents were located on a single Solaris workstation except for the agents interfacing to physical equipment (i.e. the PUAs and the SwWAs).

Figure 7.7 shows the *conceptual* location of the agents in the IMPACT system. The PUAs are located in the user's domain, whereas the CAs are located in the edge network domain. The NPAs, SPAs, and RAs are all located in the core network domain. Finally, the SwWAs are located in either the edge or the core network domain.

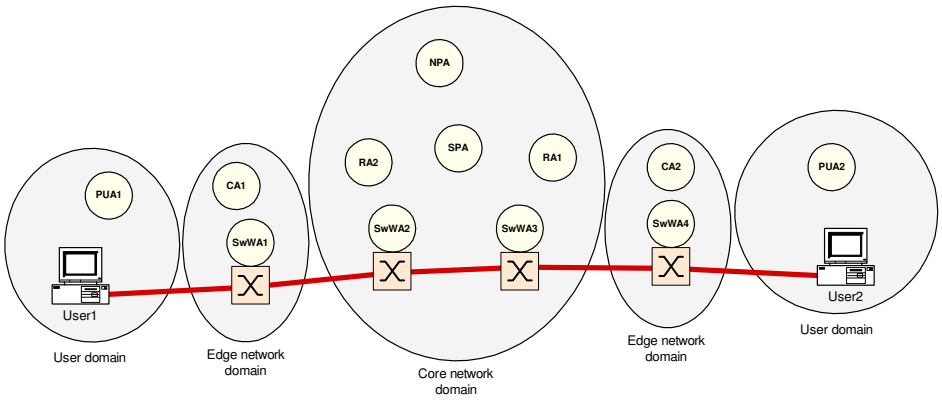


Figure 7.7 Conceptual location of the agents in the IMPACT system.

7.4 Interfacing to the ATM testbed

In order to be able to control a real ATM network, is was necessary to provide some of the agents with the ability of interfacing to the equipment. The following equipment was used in the testbed:

- PCs with Fore[2] PCA-200E ATM network cards running Linux-2.2
- Fore ASX-200 ATM switches
- Flextel ATM switches

The job of the SwWAs is to provide a common interface to the ATM switches, whereas the PUAs are responsible for interfacing to the PCs. This section describes the external interface of the PUAs and the SwWAs in more detail [83–85].

7.4.1 Interfacing to the PCs

Early on in the project it was decided to use ATM UNI signalling between the terminals and the PUAs. In this case native ATM application could be used without any modifications and IP applications could be supported using either CLIP, LANE, or MPOA on top of the ATM network [86–88].

[2]Fore was acquired by GEC (General Electric Company) in 1999. Later the same year GEC renamed to Marconi.

Protocols in the PUA

Figure 7.8 shows the protocol layers in use in the terminal and underneath the PUA. Native ATM applications can use BSD-sockets (or WinSock, if Microsoft Windows is used in the terminal) to request an ATM connection from the operating system. The operating system then sends a UNI-3.1 or an ITU-T Q.2931 *SETUP* signalling message to the control layer in the nearest ATM switch.

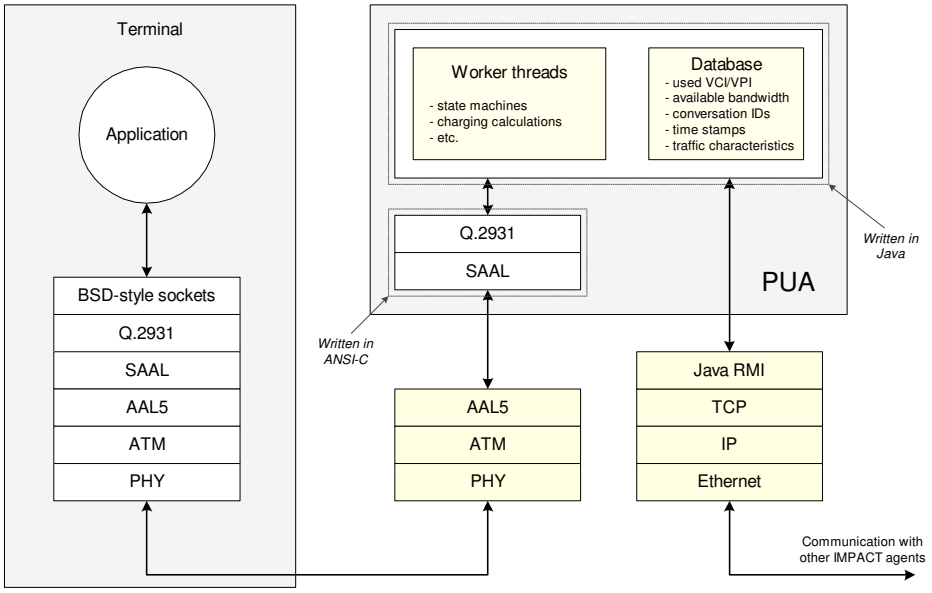


Figure 7.8 Protocols in use.

In the IMPACT system, however, the PUA intercepts the UNI signalling messages — or rather, it implements the network side of a Q.2931 signalling stack. As shown in Figure 7.8 the signalling layer and the SAAL (Signalling ATM Adaptation Layer [89]) layer are implemented in C, mainly due to stability issues with Java JNI[3]. The SAAL layer was heavily based on the SAAL implementation from the Linux-ATM project [90].

The main part of the PUA is implemented in Java, and it includes state-machines keeping track of live connections and making sure that unexpected events are handled gracefully. SDL diagrams describing the state machines in the PUA can be found in [91]. The PUA also includes a small database with the used VCI/VPI values, available bandwidth on the link from the terminal, used conversation IDs,

[3]Java Native Interface (JNI) allows Java programs to link to libraries compiled for a specific platform — in this case the Linux-ATM libraries. Early versions of JNI had some stability problems under Linux.

etc. Finally, the PUA uses Java RMI over Ethernet to exchange FIPA messages with other agents.

The chain of messages involved in successfully setting up an ATM connection from the PUA's point of view is illustrated in Figure 7.9. When the source PUA receives a *SETUP* message from the terminal it locates the destination PUA and sends a FIPA message with all information elements from the *SETUP* message. The destination PUA checks whether the destination terminal is willing to accept the connection and replies to the source PUA.

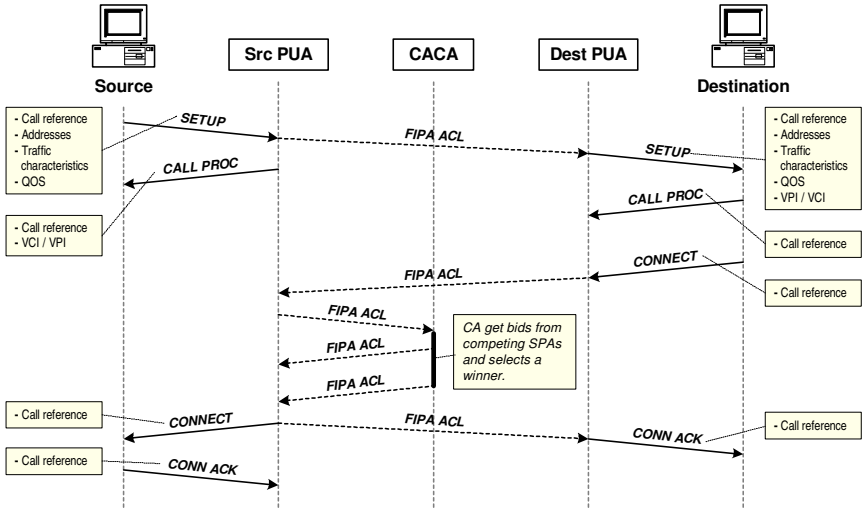


Figure 7.9 Successful connection setup.

The source PUA then passes on the connection request to the CA and the CA receives bids for the connections possibly from several competing SPAs. Based on instructions given by the PUA, the CA selects a winner and requests an RA belonging to the winning SPA to set up the connection (interaction diagrams of the communication between the agents can be found in [75]).

Finally, after the CA has informed the source PUA of a successful connection setup, the source PUA sends a *CONNECT* signalling message to the source terminal, and the destination PUA sends a *CONNECT ACKNOWLEDGE* message to the destination PUA.

Connecting the PUAs to the terminals

Unfortunately, commercial ATM switches are not designed to let custom applications integrate easily with the control software in the switch. The small ATM

switch delivered by Flextel (the Italian partner in the IMPACT project) had an open architecture based on RedHat Linux [92]. In this way the PUA could easily get direct access to a raw AAL5 connection and run directly on the Flextel switch.

In order to be able to establish an ATM network of a reasonable size it was, nevertheless, necessary to use ATM switches from other vendors as well. To do this, the signalling PVC (permanent virtual connection) was simply looped back to the terminal where the PUA was located. This is shown in Figure 7.10. Moreover, the UNI signalling PVC was configured to use VPI=0 and VCI=32 (instead of VCI=5, which is the standardized VCI for UNI signalling) because some ATM switches *always* terminate VCI=5.

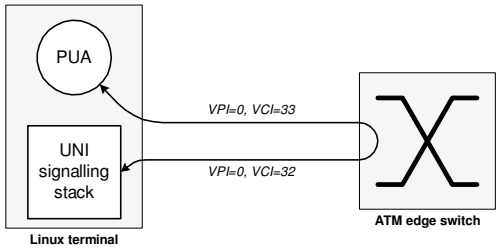


Figure 7.10 The signalling PVC is looped back to the terminal.

UNI signalling extensions

As explained in Section 7.3.6 the IMPACT system needed two extra parameters (*allow re-routing* and *allow slow connection setup*) that are not supported by legacy UNI signalling. These two parameters are passed from the calling party to the PUA through the SAP (Service Access Point) signalling information element — more specifically through the BLLI (Broadband Low-Layer Information) part of the SAP (see Figure 7.11).

7.4.2 Interfacing to the ATM switches

The IMPACT agents have access to the ATM switches by means of the switch wrapper agents. Since the API of the Flextel switches was available, the SwWAs could easily and quickly set up and tear down connections in the Flextel switches.

The other ATM switches (for instance Fore ASX-200) could not be accessed in this way. Instead the SwWAs used SNMP to access these switches. Naturally, the speed of using SNMP is considerable slower than directly calling a library function. Typically, it takes 400–600 milliseconds to set up a connection using SNMP with a Fore ASX-200 ATM switch. Of course this had a big impact on

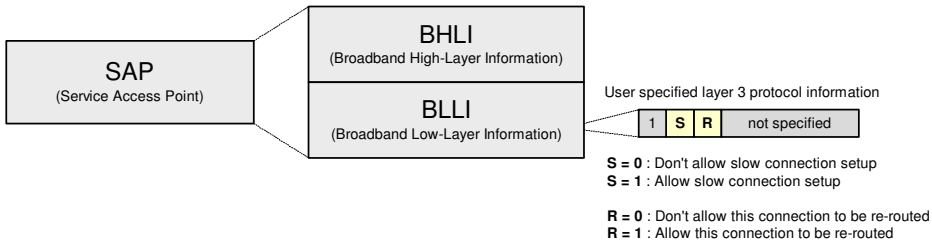


Figure 7.11 Two non-standard parameters are passed to the IMPACT agent system through the BLLI specified at connection setup by the caller.

the overall performance of the IMPACT system, however it was decided that this would still be acceptable for demonstrating the capabilities of the agent system.

7.5 Competing service providers

An important part of the flexibility of the architecture used in the IMPACT system is the ability to support selection of alternative service providers by the customers (represented by the PUA). When a user requests a connection (through the PUA) the service providers may send back an offer for the particular connection. The offers may come from service providers with whom the user has a contract *or* from service providers not even known to the user.

Typically, if the user has a contract with a service provider, the tariff structure has already been negotiated. If there is not pre-negotiated tariff structure, the price will be negotiated using one of several auction types.

7.5.1 Auction types

In general, auctions are useful when selling something of undetermined value [93]. The idea behind an auction is simple: The seller wishes to obtain as much money as possible, and the buyer want the lowest possible price. One characteristic of an auction, is that the auctioneer does not own the goods. Rather than that, the auctioneer acts as a *representative* for the seller trying to get the best price.

In many situations the buyer knows more than the seller about the value of the item. If the seller sets the price too low it can prove costly, so an auction can extract information about the value of the item not available otherwise.

Auction types can be divided into two main categories — open auctions and sealed-bid auctions. In open auctions — as the name suggests — the bidders know the offers given by other bidders and can adjust their behaviour to this information. On the other hand, in sealed-bid auctions the bidders have to make their offer without knowing anything about the other bidders. Moreover, in some auctions the price ascends and in others the price drops at regular intervals.

It is generally accepted that there are four major auction types. These are [93]:

- English auction
- Dutch auction
- First-price sealed-bid auction
- Vickrey auction

The names of the auction types used here corresponds to the names used in the *academic* world. Unfortunately, the naming convention is not unambiguous. For instance, the first-price sealed-bid auction is referred to as an English auction except in Great Britain, where it is known as the American auction [93].

English auction

The English auction (also known as "open out-cry auction") is an auction type familiar to most people. The seller announces a low opening bid, and the bidding increases until a winner is found. In some cases the seller will also announce a reserve price (lowest acceptable price) — so if the winning bid is below the reserve price, the item will not be sold at all.

One of the problems with the English auction is that it is susceptible to *rings*. A ring is a group of bidders who have an agreement with not to outbid each other. One (poor) solution to this is to keep the reserve price secret, thereby not risking selling at a too low price.

Another problem with the English auction (from the buyer's point of view) is the so-called *winner's course*. This occurs when the bidders get too enthusiastic and loose track of the actual value of the item [93].

Dutch auction

In the Dutch auction the seller announces a very high opening price. Hereafter the price is lowered at regular intervals until a bidder accepts the price. Then the auction is over and the item will be sold at this price.

If a bidder *really* wants the item, he can not afford to wait too long — otherwise it might be too late. This means that the bidder might forced to accept a price which is higher than his valuation. Due to this, the Dutch auction might be superior to the English auction, where the bidder might end up paying well below his valuation.

First-price sealed-bid auction

In contrast to the English and the Dutch auctions this auction type is sealed — i.e. the bids are kept as a secret between the bidder and the auctioneer. The first-price sealed-bid auction consists of two distinct phases. In the first phase the bidders make their bids and submit them to the auctioneer. In the second phase the bids are opened and the winner is found. The winner is the bidder who made the highest bid, and the price of the item equals the bid.

If more than one item is sold, the second item goes to the bidder offering the second-highest price and so on. This continues until all items are sold. This means that the winners will normally pay different prices for the same item.

Vickrey auction

This auction type is also called the uniform second-price auction. Like first-price sealed-bid auctions this action type is also sealed. The bidders make their bids without knowledge of the bids of the others. The winner is the bidder who offered the highest price — the price of the item, however, is the *second*-highest bid (the highest loosing price).

If more than one item is sold, the next winner is the bidder offering the second-highest price and so on. However, *all* winners still pay the same price which is the highest loosing price (hence the name "*uniform* second-price auction").

Example: Two items are offered for sale. Four bidders (A, B, C, and D) bid for the items.

A bids $32, B bids $30, C bids $25, and D bids $28.

The winners are A and B because they made the two highest bids. The price they have to pay — the highest loosing price — is $28.

One might wonder about the usability of the Vickrey action from the sellers point of view. It might seem as if bidders end up paying less than what they are actually willing to do. However, this has been shown to be untrue [93]. The bidders understand the rules and adjust their bids upwards. The price paid by the winner(s) is solely dependent on the bids given by the competitors. Also, in this auction type, the bidders do not fear the winner's curse.

Summing up the auction types

The described auction types are summed up in Table 7.3 [93]. Additional auction types exist, but they are all offshoots of the four major auction types.

Table 7.3 Characteristics of the different auction types.

Auction type	Rules
English	Seller announces reserve price or some low opening bid. Bidding increases progressively until demand falls. Winning bidder pays highest valuation. Bidder may re-assess evaluation during auction.
Dutch	Seller announces very high opening bid. Bid is lowered progressively until demand rises to match supply.
First-price sealed-bid	Bids submitted in written form with no knowledge of bids of others. Winner pays the exact amount he bid. If more than one item is sold, the winners usually pay different prices.
Vickrey	Bids submitted in written form with no knowledge of the bids of others. Winner pays the second-highest amount bid or the highest loosing price, if more than one item is for sale.

7.5.2 Selection of service provider

The implemented mechanisms for selecting a service provider for a specific connection relies on a combination of pre-negotiated tariff structures and the first-price sealed-bid auction. If a user (represented by a PUA) has a contract with a service provider, the tariff structure will normally be well-defined and fixed within each accounting period.

In an open market service providers may enter or leave the market dynamically which means that a service provider may be unknown to the user. The auction mechanism is used to have the option of selecting a service provider unknown to the user. Naturally the user will be interested in saving money, so the winner of the first-price sealed-bid auction will be the service provider bidding the *lowest* price (in contrast to the descriptions in Section 7.5.1, where the *highest* bid is the winning bid).

In this simple scenario the procedure for the selection of service provider is as follows:

1. The PUA sends instructions to the CA as a list of (SP Id, price)-pairs for the SPs with which the user has a contract. (**Note:** The user is fully aware of the price.)

2. The CA checks with those SPAs[4] having the user as a subscribed customer, to see if they can offer to set up the connection requested by the user.

3. All SPs not capable of accommodating the user's connection request are stripped from the list of (SP Id, price)-pairs.

4. The CA receives offers from SPAs unknown to the user.

5. The CA uses the first-price seal-bid auction to select a winner (the SPA bidding the lowest price) among the service providers unknown to the user.

6. This winning bid is compared to the list of prices received by the PUA.

7. The lowest price is identified and the winning SP is found.

The mechanisms for selecting service providers also supports volume-based price modifications. This means that the price per unit may decrease (or increase) if the user buys more than a threshold quantity within one accounting period. In this way the user can be rewarded by staying loyal to one service provider if he is able to predict his behaviour within this financial period. This prediction could be based on statistics gathered in the previous accounting period(s).

An example of how the tariff structures could be for two service providers is shown in Figure 7.12. SP1 is cheapest for small quantities of traffic, whereas SP2 will eventually become cheaper than SP1 for large quantities. So if the user expects to buy large quantities of capacity within this accounting period, SP2 will be the cheapest choice.

Bearing the tariff structures, the currently used quantity and expected quantity in mind the PUA calculates *adjusted prices* for all the SPs with whom there is a contract. Otherwise, the procedure described above is the same. An example of the SP selection procedure using adjusted prices is shown in Section 7.5.3.

Customising the selection mechanisms

Actually, since Java objects are used to exchange information between agents, the instructions sent from the PUA to the CA may even contain executable Java code and not only numeral parameters. In this way a PUA can completely customise

[4]The communication with the SPAs is done indirectly through the RAs.

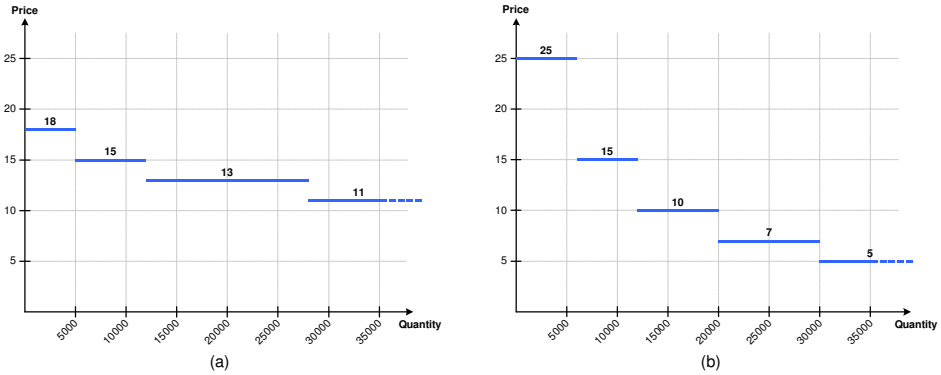


Figure 7.12 Tariff structures for two service providers: (a) SP1 and (b) SP2.

the SP selection procedure, for instance by implementing another auction method or by blacklisting certain service providers for various reasons.

Uploading executable code from one agent to another may result in security issues. So in reality the uploaded code would have to be certified by an authority trusted by all involved parties.

7.5.3 Selection of service provider — examples

Example 1

Assume for simplicity that the tariffing is based exclusively on demanded traffic volume — i.e. call distance, time of day, quality of service, etc. can be completely ignored here.

A user is subscribed to two service providers, SP1 and SP2 (represented by the agents SPA1 and SPA2, respectively). The tariff structures of these service providers are shown in Figure 7.12. Additionally, the user has already bought some units[5] from SP1 and SP2 within the current financial period – approximately 3500 units from each.

Figure 7.13 and Figure 7.14 show the charging profiles for the two service providers, SP1 and SP2. At the current usage the price per unit would be 25 when choosing SP1 whereas the price when choosing SP2 would only be 18.

However, based on statistics gathered from previous accounting periods, the user expects to use approximately 37000 units in this accounting period. Since the user has already bought a total of 7000 units, 30000 units remain. So, if the users stays

[5] One unit corresponds to one megabyte.

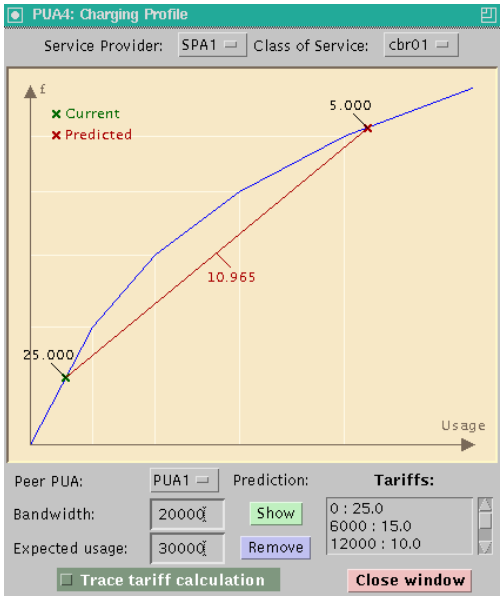


Figure 7.13 Charging profile of SP1 for a particular customer.

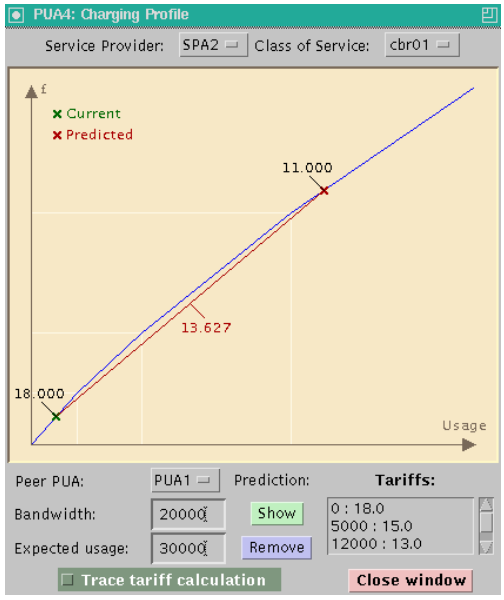


Figure 7.14 Charging profile of SP2 for a particular customer.

loyal to SP1, the average price per unit would decrease to approximately 11.0, whereas the average price per would unit would be 13.6 if staying loyal to SP2.

This means that the PUA will send the following list of (SP Id, adjusted price)-pairs to the CA when a connection request arrives from the user:

```
{(SPA1, 11.0), (SPA2, 13.6)}
```

The CA checks with SPA1 and SPA2 (and possibly with other SPAs) to see if they can accommodate the incoming connection. If no bids are received from unknown SPAs the connection will go to SP1 because it is cheaper in the long run.

Example 2

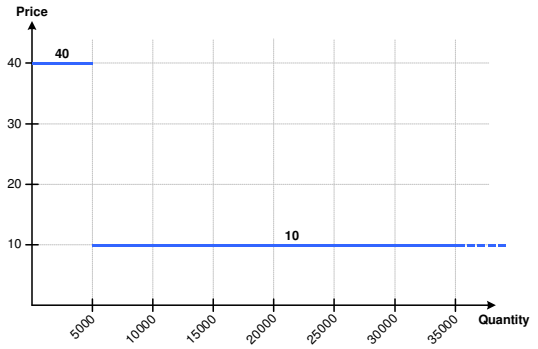
In this case an SPA unknown to the user bids for the connection. If the bid is lower than 11.0 the PUA will select this unknown SPA. Otherwise — if the bid is higher than 11.0 — the connection will still go to SPA1.

7.5.4 Drawbacks

It is tremendously important that the procedures for selecting the service provider remains secret between the PUA and the CA. If a service provider knows the details about the selection procedure he can quite easily take advantage of this and use it to squeeze extra money out of the user. This problem can, however, be reduced if different PUAs use non-identical selection procedures.

The following example illustrates how a service provider can misuse information about how the user selects the service provider.

Example: User A has a contract with the service provider SP1. The user and SP1 agree upon the following tariff structure:



User A knows that the initial price per unit (which is 40) is pretty high compared to other service providers, but since he expects to buy 30000 unites within each financial period the average price per unit would be 15, which is cheaper than the competitors.

If user A stays loyal to SP1 *and* if the user indeed buys 30000 units within one accounting period, the total price would be:

$$\text{Total price: } 5000 \times 40 + 25000 \times 10 = 450000$$
$$\text{Average price per unit: } \frac{450000}{30000} = 15$$

Now, suppose that the service provider SP1 owns another service provider, SP2. Since SP1 has had user A as his customer for several financial periods he has gathered a lot of statistical data about the user's behaviour — this means that he knows that the user *expects* that the average price per unit will be 15. Also, suppose that no SPs have been blacklisted by user A and that the user buys 30000 units within this financial period.

In the beginning SP2 bids for the connections. The price per unit offered by SP2 is 14 (i.e. a little bit lower than the expected average price per unit) — i.e. this is lower than the expected average price per unit offered by SP1. Therefore the user chooses SP2 for setting up the first few connections. User A has no reason to believe that SP2 will change his behaviour, so it seems reasonable to go for SP2 instead of SP1.

However, once user A has bought 10000 units SP2 stops bidding for the connections for the remaining part of the financial period. This means that the user is left back with SP1 for the remaining 20000 units in this financial period. In this case the total price and the average price per unit is:

$$\text{Total price: } 10000 \times 14 + 5000 \times 40 + 15000 \times 10 = 490000$$

$$\text{Average price per unit: } \frac{490000}{30000} = 16.3$$

It could become even worse if SP2 keeps bidding until user A has bought 25000 units. In this situation the user might end up buying the first 25000 units from SP2 and the remaining 5000 (expensive) units from SP1. However, since the average price per unit expected by the user for the last 5000 units is 40, it is very possible that other service providers (not owned by SP1) will bid lower than 40 and thereby win the auction. Of course SP1 will do anything possible to avoid this.

The previous example illustrates some problems that could occur when users entrust computer software (in this case, software agents) with delicate decisions. The problem is not so predominant when the decisions are made by real human beings, since they normally behave in different and unpredictable ways.

7.6 Virtual path selection strategies

When a user requests a connection, the first task is selecting which service provider to use for the connection (see Section 7.5.2 and Section 7.5.3). If several alternative paths exist — i.e. the RA owned by the selected service provider controls several virtual paths capable of accommodating the connection request — the RA can choose between different strategies for placing connection. Possible strategies for placing connections could be as follows:

- Maximise the minimum residual capacity
- Maximise the maximum residual capacity
- Minimise hop count

There is no clear winner among these strategies — pros and cons exist for all of them. The characteristics of the strategies will be described in the remaining part of this section, and some small examples will be given for illustration purposes.

7.6.1 Maximise the minimum residual capacity

Actually, *maximising the minimum residual* capacity is another word for balancing the load across the possible virtual paths. However, the used term is more precise with respect to the mechanisms used to balance the load. If an RA controls more than one path from a given source to a given destination it places the call in such a way that the minimum remaining (residual) capacity is as large as possible.

Example: Suppose that an RA controls three virtual paths between a source-destination pair. A user requests a connection between this s-d pair with a bandwidth of 20000 cps[6], which is lower than the capacity of each of the three virtual paths. Before the connection is set up the capacities of the three VPs (VP1, VP2, and VP3) are:

No connections

VP name	*Bandwidth*
VP1	30000
VP2	40000
VP3	50000

The RA examines the residual capacities of the three VPs and finds out that the requested connection can be placed in any of them. Placing the requested connection in VP1, VP2, or VP3, respectively, results in the following residual bandwidths for the three VPs:

VP1 is used	
VP1	**10000** ←
VP2	40000
VP3	50000

VP2 is used	
VP1	30000
VP2	**20000** ←
VP3	50000

VP3 is used	
VP1	**30000** ←
VP2	40000
VP3	30000

In all three cases, the minimum residual capacity is marked with an arrow (←). From this it is clear that the maximum minimum residual is obtained if the connection is placed in VP3 — i.e. when using the *maximise-the-minimum-residual* strategy, the connection will be placed in VP3.

When placing the 20000 cps connection in VP3 the RA is still capable of accepting 3 more calls with a bandwidth of 30000 cps (the maximum minimum residual bandwidth). However, if a user requests a 50000 cps connection between the s-d pair controlled by this RA, the connection has to be rejected (unless the first connection allows re-routing).

On the other hand, *if* VP1 or VP2 were chosen for the 20000 cps connection, there would only be capacity left for 2 calls with a bandwidth of 30000 cps.

From the example it is clear that this strategy has some advantages and some drawbacks. High-bandwidth connections are sacrificed in order to be able to allow more low-bandwidth connections to be routed through the VPs.

7.6.2 Maximise the maximum residual capacity

Maximising the maximum residual capacity is the opposite of maximising the minimum residual capacity (see Section 7.6.1). Put in another way, if an RA has more than one option for placing an incoming call, it will place the call so that the maximum residual capacity among the VPs becomes as big as possible. The following example illustrates the mechanism behind this strategy. In addition, the example shows how the risk of ambiguity can be reduced by calculating the sum of the residual capacity squared.

Example: Once more, suppose that an RA controls three VPs between an s-d pair. Again a user requests a connection between this s-d pair with a bandwidth of 20000 cps, and again this is lower than the capacity of each of the three virtual paths. Like before the capacities of the VPs before setting up the connection are:

[6] ATM cells per second (cps).

No connections	
VP name	*Bandwidth*
VP1	30000
VP2	40000
VP3	50000

Placing the requested connection in VP1, VP2, or VP3, respectively, results in the following residual bandwidths for the three VPs:

VP1 is used	
VP1	10000
VP2	40000
VP3	**50000** ←

VP2 is used	
VP1	30000
VP2	20000
VP3	**50000** ←

VP3 is used	
VP1	30000
VP2	**40000** ←
VP3	30000

In all three cases, the *maximum* residual capacity is marked with an arrow (←). In this case the maximum residual bandwidth is 50000 cps, and it can be obtained by using either VP1 or VP2 for the connection. This means that the requested connection will be placed in either VP1 or VP2.

If the 20000 cps connection is placed in VP1, the RA can accept *two* high-bandwidth connections (40000 cps and 50000 cps). On the other hand, if the connection is placed in VP2, the RA will have room left for *one* high-bandwidth connection and two medium-bandwidth connections (20000 cps and 30000 cps).

The risk of ambiguity can be reduced by calculating the sum of the residual capacities squared, and then choose the VP where the resulting number is as big as possible. Putting in the numbers from this example gives the following:

$$\begin{array}{lrcl} \text{VP1 is used:} & 10000^2 + 40000^2 + 50000^2 & = & 4,200 \times 10^6 \\ \text{VP2 is used:} & 30000^2 + 20000^2 + 50000^2 & = & 3,800 \times 10^6 \\ \text{VP3 is used:} & 30000^2 + 40000^2 + 30000^2 & = & 3,400 \times 10^6 \end{array}$$

Using these calculations as the guideline for choosing the VP to use for the connection tell us that the connection will go to VP1[7].

As expected, the *maximise-the-maximum-residual* strategy favours high-bandwidth connections.

[7] The calculations *could* also be used for the *maximise-the-minimum-residual* strategy. However, of course the VP resulting in the smallest number must be used. Also, this method does *not* reduce the risk of ambiguity for the *maximise-the-minimum-residual* strategy.

7.6.3 Minimise hop count

In addition to the two previous strategies, the RA can also choose to place the incoming connections so that the VP with the lowest hop-count is filled first, and then the VP with the second-lowest hop-count, and so forth. Actually, the RA has knowledge about the actual routes traversed by the VPs. This knowledge is crucial in order to be able to transfer capacity between VPs controlled by different RAs.

Since it is assumed that the VPs between each source-destination pair has *approximately* the same hop-count, minimising the hop-count will only have a very limited impact. Therefore this strategy is not considered in the remaining part of this chapter.

7.6.4 Pros and cons

Even though the two strategies for placing connections described in the Sections 7.6.1 and 7.6.2 are opposites of each other it is not possible to declare a clear winner among the two. The best strategy to use must rely on the expected behaviour by the users. If the service provider expects that the users will demand high-bandwidth connections the best strategy will of course be the *maximise-the-maximum-residual* strategy and vice versa. Both strategies were considered in the IMPACT project, however the *maximise-the-minimum-residual* strategy was used for the live demonstration of the project.

7.7 Capacity and connection transfer scenario

This section describes a scenario demonstrating some of the features of the IMPACT agent system. This scenario has also been demonstrated in a real system with five ATM switches.

Consider the network shown in Figure 7.15. Only the relevant PCs (PC1, PC2, and PC3) and VPs are shown here. Additionally, the VPs and the physical links from the edge switches to the users have not been included since they are of no interest to this scenario.

PC1 can reach PC2 through three alternative VPs — VP1, VP2, or VP3. PC2 and PC3, however, can only use VP4 for interconnections. VP1, VP2, and VP3 are controlled by the resource agent RA1, whereas VP4 is controlled by RA2.

If several virtual paths exist between a source-destination pair the RA can choose between different strategies when deciding which VP to use for the next connection. The RAs in this scenario are trying to *maximise the minimum residual* bandwidth when setting up new connections.

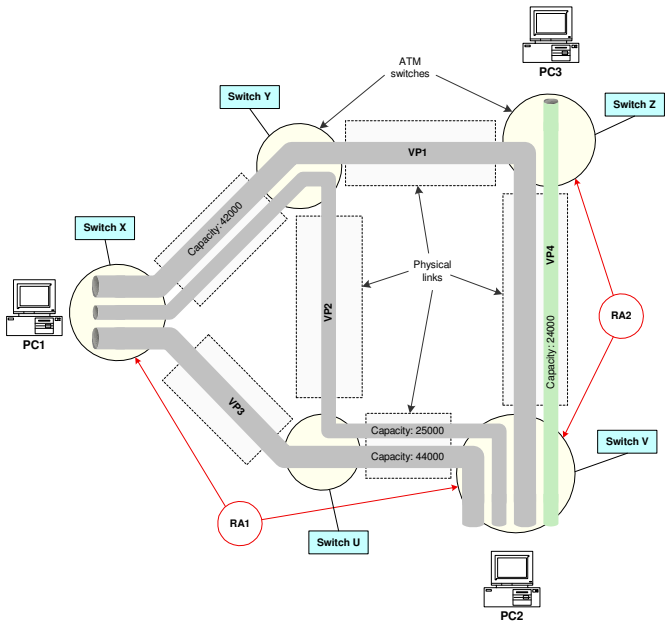


Figure 7.15 Three PCs connected to an ATM network with five switches.

Figure 7.16 shows the situation after 4 virtual channels with a bandwidth of 20000 have been set up — three connections between PC1 and PC2 and one connection between PC3 and PC2. Due to the VP selection strategy the three connections between PC1 and PC2 have been assigned to different VPs (VP1, VP2, and VP3). The connection from PC3 to PC2, on the other hand, *must* use VP4. The residual capacities of the four VPs are also shown in the figure.

If PC3 requests a connection to PC2 with a bandwidth of 20000 it is clear from Figure 7.16 that the connection will be refused, since the residual capacity is only 4000. However, *if* PC3 sets the *allow-slow-connection-setup* bit in the *SETUP* signalling message, RA2 will use some more time to make an extra effort finding enough capacity for the connection.

First, RA2 will try to see if *capacity transfer* can bring about enough capacity for the new connection. RA2 asks the other RAs whether they own a VP sharing some of its route with VP4 *and* whether they can spare 16000 units of bandwidth (with 16000 extra units of bandwidth, the total spare capacity of VP4 would sum up to 20000). RA1 replies with a positive response telling that RA2 can borrow 16000 units from VP1. The situation *after* transferring 16000 units of capacity from VP1 to VP4 is shown in Figure 7.17.

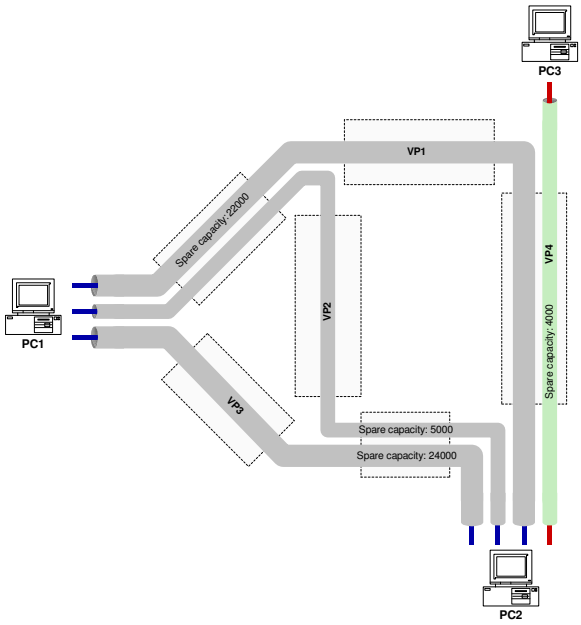


Figure 7.16 The situation after 4 virtual connections with a bandwidth of 20000 each have been set up.

After the capacity transfer, RA2 can easily place the new connection requested by PC3 in VP4. This is shown in Figure 7.18.

Now, suppose that PC3 requests a *third* connection to PC2[8]. Once again, RA2 asks RA1 if 20000 units of capacity can be spared. This time, however, the residual capacity of VP1 is only 6000. Therefore, RA1 moves on to the next possible solution to the problem: *Connection transfer*.

RA1 figures out, that VP3 has a residual capacity of 24000, so the connection in VP1 with a bandwidth of 20000 *may* be moved to VP3. Before doing this, RA1 checks whether the connection in VP1 allows re-routing (i.e. whether the *allow re-routing* bit was set in the original *SETUP* signalling message).

RA1 finds out, that the connection in VP1 does indeed allow re-routing, so the connection is moved to VP3 causing a connection downtime of approximately 3 seconds. The situation after a connection has been moved from VP1 to VP3 is shown in Figure 7.19.

Now, the residual capacity of VP1 is 26000 units, so 20000 units can easily be spared. Again, RA2 borrows 20000 units of capacity from RA1 and assigns it to

[8]Again, the *allow-slow-connection-setup* bit is set in the *SETUP* signalling message.

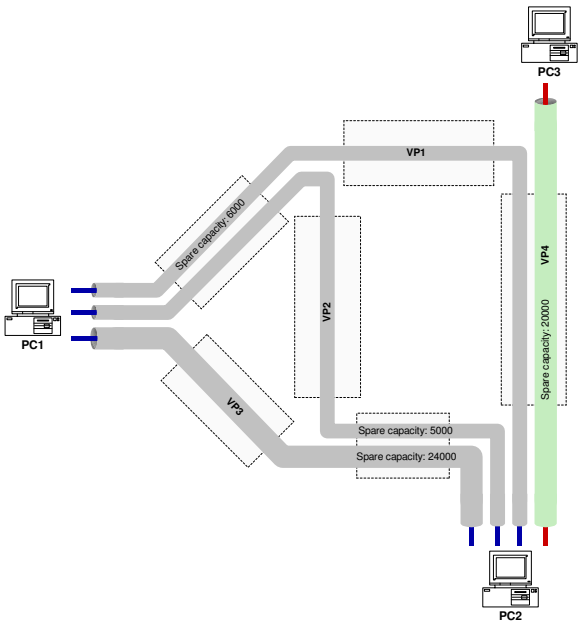


Figure 7.17 The situation after 16000 units of capacity has been transferred from VP1 to VP4.

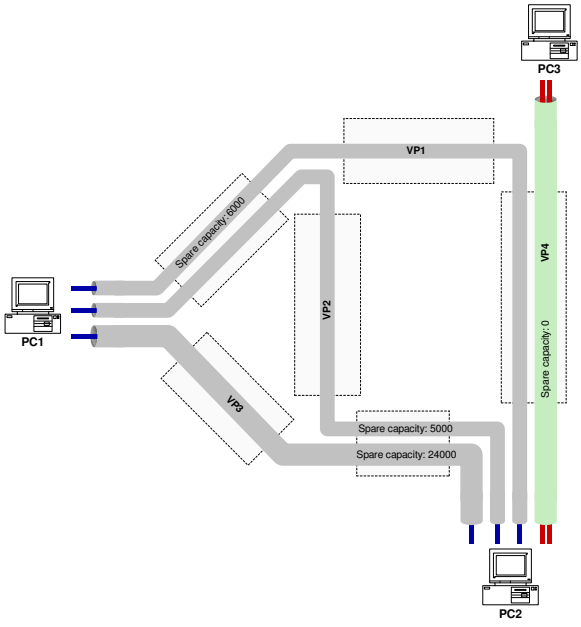


Figure 7.18 Another connection with a bandwidth of 20000 is placed in VP4.

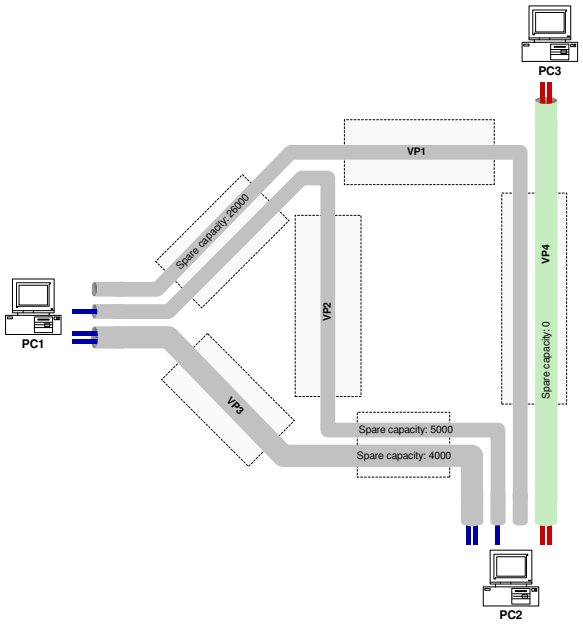


Figure 7.19 The connection in VP1 is moved to VP3.

VP4, so suddenly the residual capacity of VP4 is 20000 units — exactly what is needed for the third connection. This is depicted in Figure 7.20.

The third connection from PC3 to PC2 is then placed in VP4. Figure 7.21 shows the situation after all three connections from PC3 to PC2.

This scenario has shown how the agent system is capable of communicating with each other in search of spare capacity in order to be able to carry more connections. Without the negotiating agents, only 4 connection (with a bandwidth of 20000 unit) would have been possible, but the capacity transfer and connection transfer makes it possible to add 2 extra connections.

Obviously, when RA1 lends 40000 units of capacity to RA2 it is only used between switch Z and switch V (see Figure 7.15). Therefore the capacity should be transferred back as quickly as possible *or* 40000 units of capacity between switch X and switch Y as well as 40000 units of capacity between switch Y and switch Z could be lend to other resource agents in need of capacity.

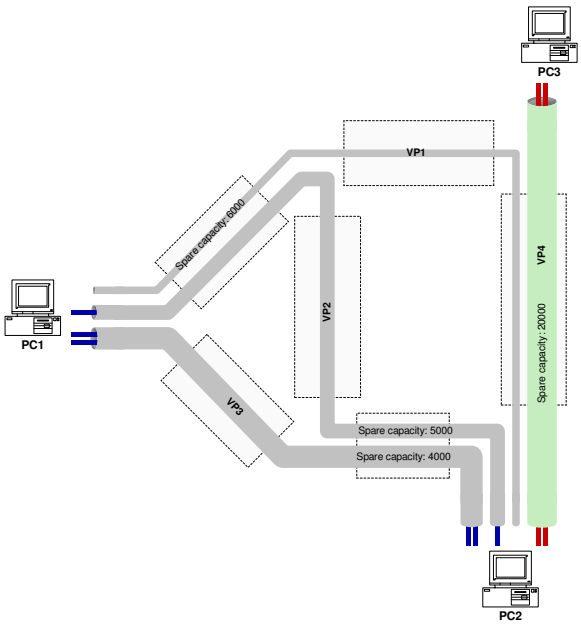


Figure 7.20 The situation after 20000 units of bandwidth has been transferred from VP1 to VP4.

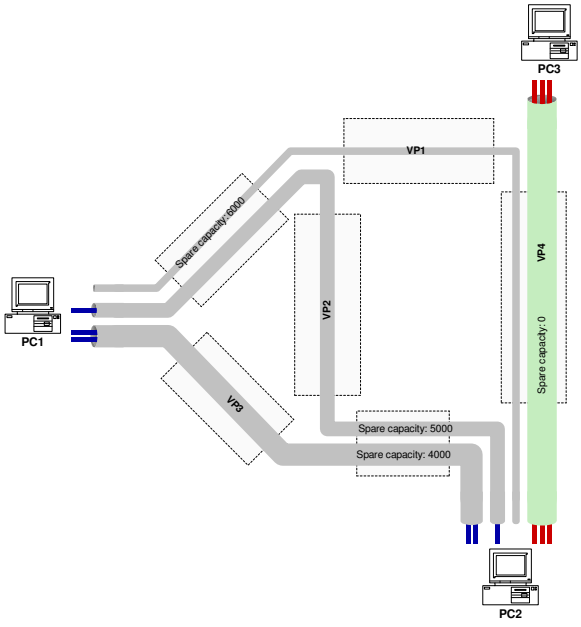


Figure 7.21 A third connection with a bandwidth of 20000 is placed in VP4.

7.8 Performance

Even though it was decided early on in the project that the speed of Java would be (or *had to be*) adequate for IMPACT purposes, it would still be useful to measure whether connection setup delays could stay within reasonable boundaries.

Figure 7.22 shows the physical setup used to make performance measurements. A total of 22 IMPACT agents were run on a single Solaris-based dual-Pentium III server in order to make a worst-case scenario.

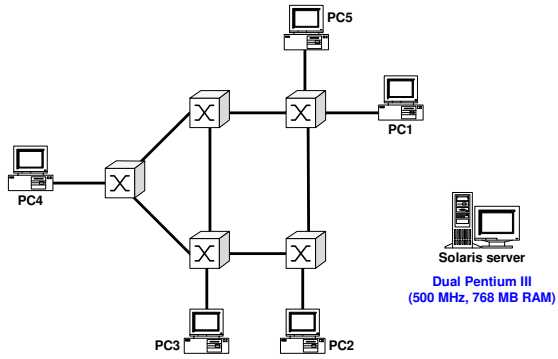


Figure 7.22 Performance measurement setup.

7.8.1 Expectations

In the best-case scenario the connection set-up time will consist of the time it takes to execute the Java byte-code in the different agents plus the time it takes to generate and parse the ATM UNI signalling messages. If there were no such thing as scheduling delays in the operating system or network congestion, this minimum amount of time would be almost constant.

However, due to network congestion (which in turn might call for retransmission of ACL messages), OS scheduling, swapping to the hard-drive, hardware interrupts, etc., the connection set-up time will not remain constant. If these variations are assumed to be completely random, the probability of a call setup being completed within a fixed time-slot beyond the constants of the system can be modelled by a binomial distribution function. If the number of connection setups is large the Poisson distribution can be used instead [94].

$$P\{X = k\} = \frac{\lambda^k}{k!} e^{-\lambda}\ ,\ k \in \mathrm{N}_0 \tag{7.1}$$

For comparison reasons, plots of the Poisson distribution for three different values are shown in Figure C.1 (Appendix C).

7.8.2 Results

Figures showing the performance of the IMPACT system are shown in Appendix C. The X-axis shows the setup time (in milliseconds), whereas the Y-axis shows the number of occurrences. Note that the time resolution is not the same in all figures [78].

Dummy SwWAs

The performance measurements of the IMPACT system when using dummy switch wrappers are shown in the Figures C.2, C.3, C.4 C.5, and C.6. Loop-back connections seem to be completed a bit faster than point-to-point and local connections, which was expected because loop-back connections are treated differently within the agents — especially within the PUAs. Connection setups requiring capacity transfer were only slightly slower, whereas connection setups requiring re-routing of other connections were 100-200 milliseconds slower.

Real SwWAs

The Figures C.7, C.8, C.9, C.10, and C.11 show the measurements using real switch wrappers and real ATM switches (Fore-ASX200). Again loop-back connections are a bit faster than point-to-point and local connections. The influence of capacity transfer, however, is almost negligible. Connections requiring re-routing are by far the slowest to set up. From the measurements with the dummy SwWAs it is clear the the big difference is caused by the time it takes to control the Fore-ASX200 switches through SNMP (first delete two VCs in the edge switches, then re-create the VCs, and finally set up the new connection).

Legacy ATM UNI signalling (Fore-ASX200)

Measurements using UNI-3.1 signalling of the Fore-ASX200 switches are presented in the Figures C.12, and C.13. It is quite obvious that ATM UNI signalling outperforms the IMPACT system. However, these performance measurements can not be compared directly to the measurements of the IMPACT system, since they do not include any NNI signalling.

Comparison

The average setup time and the standard deviation of all connection setup time measurements are shown in the Tables 7.4, 7.5, and 7.6.

Table 7.4 Average connection setup times using dummy SwWAs (in milliseconds).

	Local connections	Loop-back connections	P-to-P connections	Capacity transfer	Re-routing
Average	555	393	413	451	612
Std. deviation	153	119	92	87	114

Table 7.5 Average connection setup times using "real" SwWAs (in milliseconds).

	Local connections	Loop-back connections	P-to-P connections	Capacity transfer	Re-routing
Average	1083	913	933	982	2674
Std. deviation	189	164	137	131	445

Table 7.6 Average connection setup times using UNI signalling (in milliseconds).

	Local connections	Loop-back connections
Average	29	62
Std. deviation	11	80

When using real SwWAs instead of dummy SwWAs (except for the re-routing case) connections use approximately 0.5 seconds extra to be set up. It is therefore quite obvious, that setting up a connection using SNMP takes in the order of 500 milliseconds.

The influence of using real SwWAs is particularly visible for the re-routing scenario. In this case the switch operations (again using SNMP) add a penalty of approximately 2 seconds. These 2 seconds correspond to the time at takes to remove a connection and restore it again along another route, and the time it takes to set up the new connection.

All in all, these measurements have shown that the speed of Java is not the weakest link. Even though something could be gained from using C or C++, the major

contributor to the setup delay is SNMP. In addition, Java has become considerably faster through the latest releases from Sun Microsystems.

7.9 Limitations and workarounds

Obviously, the IMPACT system has some problems and drawbacks, that ought to have been addressed, if time had allowed so. This section runs through the most serious problems and gives some suggestions about how to overcome some of the problems.

7.9.1 Scalability

Since each resource agent controls the VPs between a single source-destination pair, the total number of RAs in a network with n entry points[9] would be $\frac{n(n-1)}{2} \in O(n^2)$. This poses an upper limit to how large a network controlled by IMPACT agents could grow in reality.

Probably the easiest way of getting around this problem would have been by implementing the proxy connection agent (PCA) that would make is possible to communicate and set up connections traversing several IMPACT controlled domains.

7.9.2 Bandwidth utilization

In the IMPACT system, bandwidth is split up into rather small pieces handled by separate RAs. So, since each RA has no global overview over network resource, it is almost impossible for the agents to allocate resource in an optimal way. The RAs will tend to find solutions that are optimal given their local view of the network, however, this will generally *not* lead to the best global solution.

7.9.3 Failure resistance

Since the software agents in the IMPACT project run as independent processes, the system can easily work even though some agents are temporarily unavailable[10]. However, if an agent is reset, it must be able to regain its states before it was reset.

The IMPACT system has *one* single-point-of-failure issue — the *directory facilitator*. If the directory facilitator is not working, the agents will not be capable

[9] Several end-systems may share one entry point.

[10] Of course, a failing agent may cause local problems. For instance, if a user's PUA is not working, that user will not be able to initiate or accept connection while the PUA is unavailable.

of finding each other, so the support for back-up directory facilitators will be of utmost importance in production systems.

7.9.4 Configuration

Well-implemented agents should be capable of figuring out their role in their society without human interaction or with very limited human interaction. The IMPACT agents, however, need to be configured specifically for each network they control. For instance, the number of VPs in the network and their routes must be configured manually.

In order to be able to use the system in large networks, it would be necessary to implement auto-configuration mechanisms. This could be done by using PNNI-like link-state flooding mechanisms to give the SPAs an overview of the states of the links, and by implementing routing algorithms in the SPAs allowing them to set up VPs by themselves, and not relying on manually configured VPs.

7.10 Benefits with agents

The work carried out by the IMPACT project revealed the following benefits of using agents:

Negotiation The project demonstrated how the resource agents can negotiate with each other and borrow bandwidth, which — in turn — can make it possible to accept connections that would otherwise be rejected.

Planning The resource agents also contain a planning layer (not described in this chapter). Using this layer, along with knowledge of previous traffic patterns, the agents were in some situations capable of dealing with high-load situations before they occurred.

Development time The whole concept of agency allowed for nice and clean interfaces between logically different functions in the network. This decreased development time and made misunderstandings a rare event.

Other benefits could have been shown as well, if it wasn't for the time constraints of the project. For instance, robustness is normally one of the key advantages of using distributed approaches. However, there was not enough time for the IMPACT project to pay special attention to this issue (see also Section 7.9.3).

7.11 Summary

This chapter has given an overview of the IMPACT project. Many things regarding the internals of each agent have been left out, however more details can be found in the project's deliverable documents [72–79].

The IMPACT project implemented a society of software agents capable of controlling and managing many aspects of an ATM network (although some shortcuts had to be made due to time constraints). The role of some agents was mainly to interface to the ATM equipment, whereas other agents carried out higher level functions such as controlling the resources of VPs, bidding for connections, etc. The role of the different types of IMPACT agents has been chosen in a way that reflects a possible open and deregulated telecommunication market, where there are several service providers.

All of the things implemented in the IMPACT project *could* have been implemented using traditional techniques. However, the use of the agent concepts and the JAM agent framework certainly made the implementation a lot easier. The development of the different agents were done in different locations (mainly London, Athens, and Copenhagen) and by using agent concepts a lot of confusion was avoided and the interoperability of the agents developed by different people turned out to cause almost no problems.

Chapter 8

Artificial Ants for Routing and Resource Management

A thorough investigation of artificial ants and their use for routing and resource management in communication networks (mainly connection-oriented) is given in this chapter. The base model used corresponds to the model described in [42], however by introducing a new class of ants — the so-called *smart ants* — some improvements can be made which add more dynamics to the network and lower bandwidth consumption from the ants. The term "smart agents" (where the agents actually behave similar to artificial ants) has already been used by E. Bonabeau et al. [41] to describe agents capable of remembering the intermediate nodes they have visited and the time they visited those nodes. In this chapter, however, the term "smart ants" is used to describe a whole *class* of artificial ants comprising 7 types of ants — some better than others.

In order to get a better idea of the usefulness of artificial ants a huge number of simulations have been made on networks with many nodes (switches or routers) providing answers to a lot of questions like the following:

- How long time does it take for the system to reach steady state?
- How much bandwidth do the artificial ants consume?
- How does the model cope with different network topologies (mesh, ring, tree)?
- What happens if the system is pushed seriously out of balance?
- How well does it scale?

- How long time does it take to recover from link failures?
- How long time does it take to discover new links?
- Is it possible to support QoS routing with artificial ants?
- How does load-balancing work with artificial ants?
- How stable is the system?

One of the main goals of this work has been to produce qualitative and *quantitative* results rather than the mainly qualitative results found in the literature. For instance, the simulation time is specified in seconds (or milliseconds) rather than "simulation cycles" in order to give a clear idea of the time-scales involved. Also, to make things more realistic the simulations in this chapter take into account the time it takes for an ant to travel from node to node. The model in [42], on the other hand, simply assumes that it takes one time-step to travel from one node to another.

Also, *if* artificial ants are ever going to be deployed in real networks, the theoretically best solution will not necessarily be the one that is going to be used. Other things need to be taken into account like scalability, needed processing power in the nodes, the size in bytes of the individual artificial ants (depending on the protocol they are going to be used with), etc.

8.1 The model

The principles of the model used to simulate the artificial ants are described in this section. This model is based on the model suggested in [42], however a lot of enhancements are added in order to obtain faster convergence and lower bandwidth consumption (by the ants). Furthermore, special care has been taken to assure that the model is actually possible to implement in a real network without causing too many changes or requiring excessive computation power.

As described in Section 4.3 ants in nature deposit pheromones as they move around in their environment. Like everything else in the nature, pheromones consist of a chemical substance which will evaporate slowly over time. This means that the strength of the trails between the ants' nest and their food sources depend on how busy the trail has been over a reasonable amount of time.

In a communication network the links between nodes (switches, routers, add/drop multiplexers, etc.) play the role of the trails in nature. However, network links are (normally) passive elements without any kind of memory. This means that the task of keeping track of the strength of the trails has to be kept within the network nodes.

8.1.1 Pheromone adjustments

As proposed in [42] each network node has a table keeping track of the strength of the pheromones. Instead of using absolute numbers, the pheromone strengths are represented by probabilities of choosing a specific output port when sending an ant to a specific destination. An example of a pheromone table for a node with three input/output ports in a network with 5 nodes is given in Figure 8.1.

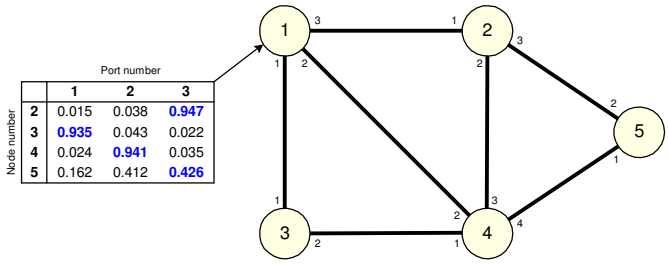


Figure 8.1 Example of a pheromone table for node 1.

According to this pheromone table, the probability of using output port 1 when sending an ant to node 3 is 93.5%, whereas ants going to node 5 will have an almost equal probability of choosing either port 2 or port 3.

The initial values in the pheromone table (before any artificial ants have been sent through the network) corresponds to the situation in the nature where the ants have no knowledge of any trail being better than others. This means that all values in the pheromone tables in each node must be equal when the simulation is reset.

Table 8.1 shows a generic pheromone table for a node in an artificial ant controlled network. The table has m columns, where m is the number of ports of the node, and $n-1$ rows, where n is the number of nodes in the network (a node does not need a row for its own node number).

Table 8.1 Generic pheromone table for one network node.

Node number	Port number				
	1	2	3	$\cdots$	m
1	$p(1,1)$	$p(1,2)$	$p(1,3)$	$\cdots$	$p(1,m)$
2	$p(2,1)$	$p(2,2)$	$p(2,3)$	$\cdots$	$p(2,m)$
3	$p(3,1)$	$p(3,2)$	$p(3,3)$	$\cdots$	$p(3,m)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	$p(n,1)$	$p(n,2)$	$p(n,3)$	$\cdots$	$p(n,m)$

In order to make sure that the entries in a pheromone table can be interpreted as probabilities the sum of all values in each row must be 1. So the following must be true for node u:

$$0 \leq p_u(i,j) \leq 1, \ \forall \ i \text{ and } j \tag{8.1}$$

and

$$\sum_{j=1}^{m} p_u(i,j) = 1 \ , \ \forall \ i \neq u \tag{8.2}$$

$$(1 \leq i \leq n \text{ and } 1 \leq j \leq m)$$

Each time an artificial ant arrives at a node, the values in the row corresponding to the node where this particular ant was created are updated in order to reflect the pheromone changes in the nature. As suggested in [42] the entries in the pheromone table are adjusted according to the following formula:

$$p(u,j) = \begin{cases} \dfrac{p_{old}(u,j) + \Delta p}{1 + \Delta p} & , \text{ if } j = v \\ \dfrac{p_{old}(u,j)}{1 + \Delta p} & , \ \forall \ j \neq v \end{cases} \tag{8.3}$$

where u is the number of the node where the ant was created, and v is the port number where the ant entered this node. Δp is a measure for the probability increase for port v and the probability decrease for all other ports (Δp depends on the age of an ant — see Section 8.1.4). After all values of row u in a pheromone table have been adjusted, the sum of the new values is:

$$
\begin{aligned}
& p(u,1) + p(u,2) + p(u,3) + \cdots + p(u,v) + \cdots + p(u,m) \\
&= \frac{p_{old}(u,1)}{1+\Delta p} + \frac{p_{old}(u,2)}{1+\Delta p} + \cdots + \frac{p_{old}(u,v)+\Delta p}{1+\Delta p} + \cdots + \frac{p_{old}(u,m)}{1+\Delta p} \\
&= \frac{\Delta p + \sum_{j=1}^{m} p_{old}(u,j)}{1+\Delta p} = \frac{1+\Delta p}{1+\Delta p} = 1
\end{aligned}
$$

Since the values in row u still sum up to 1 they can — once again — be interpreted as probabilities.

8.1.2 The life cycle of an artificial ant

All nodes in the network have the capability of creating artificial ants. At each simulation step each node has a small (configurable) probability of creating an ant. In the model used in this chapter a simple artificial ant must contain the following information:

- A special tag ("I am an ant")
- The creation time
- The source node
- The destination node

The special tag is important so that the network nodes can tell the difference between an artificial ant and a "normal" data packet (or cell). The destination node is chosen randomly among all other nodes in the network.

After the creation[1] of an artificial ant, the node examines the row in the pheromone table corresponding to the *destination* node. Based on the probabilities in this row, an output port is chosen. The higher the probability of a port, the more ants are sent out of that port.

When a node *receives* an artificial ant, it updates the row of the pheromone table corresponding to the node where the ant was created, and selects an output port based on the probabilities in the row of the pheromone table corresponding to the ant's *destination* node (just like the behaviour of the source node).

Finally, when the ant arrives at its destination it is killed. This means, that the trail produced by an artificial ant goes in the *opposite* direction of the ant's direction of motion.

[1] The rate by which artificial ants are created — the *birth rate* — is an adjustable parameter.

8.1.3 The basic model exemplified

This section explains the mechanisms described in the Sections 8.1.1 and 8.1.2 using the example network with 5 nodes shown in Figure 8.2. Before any artificial ant has travelled through the network the probabilities in the pheromone tables in each node are equal. The amount, by which the values are increased (or decreased) in this example, is irrelevant at this point. The only important thing is that one value in a row is increased, whereas the other values are decreased. Also, the values marked with the colour blue, are the biggest values in their row.

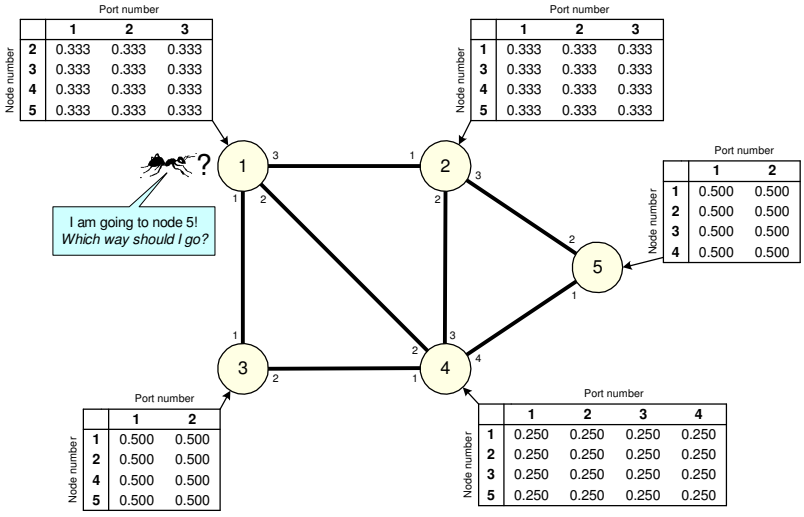


Figure 8.2 Initial values of the pheromone tables.

At some moment, node 1 creates an artificial ant with node 5 as its destination. Since the values in row 5 of node 1's pheromone table are all equal, node 1 chooses a completely random output port for the ant. In this example node 1 chooses output port 2, so the next hop on the ant's trail is node 4.

Node 4 updates its pheromone tables according to the Formula 8.3 and selects an output port for the artificial ant based on the probabilities in its own pheromone table. Node 4 chooses port 3, so this time the ant goes to node number 2, which in turn, updates its own pheromone table and passes on the artificial ant through port number 3. Finally, the artificial ant reaches node number 5 where it is killed. The consequences of this sequence of events is illustrated in Figure 8.3.

The probabilities in the pheromone tables in the nodes 2, 4, and 5 have been altered according to the trail chosen by the artificial ant. This means that if node 5 wants to send an ant to node 1, there is an increased probability, that the ant will follow

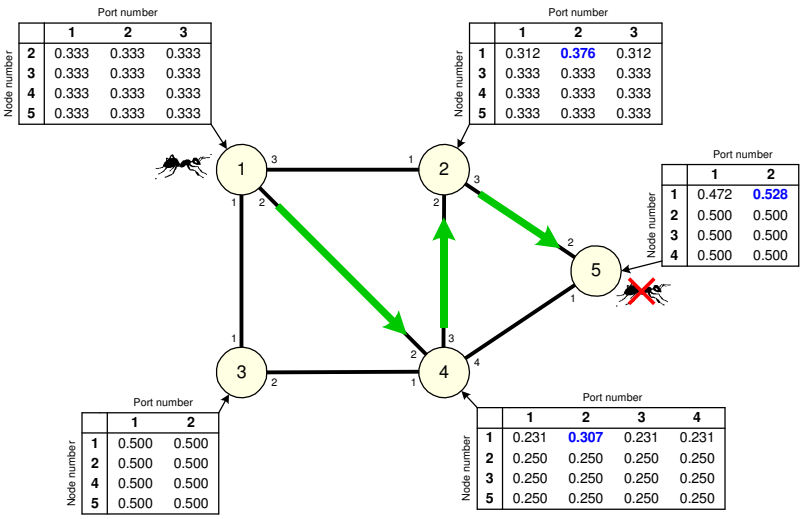


Figure 8.3 The pheromone tables after one ant has made it from node 1 to node 5.

this route:

$$\text{node } 5 \rightarrow \text{node } 2 \rightarrow \text{node } 4 \rightarrow \text{node } 1$$

The simple example network has been examined in the simulator[2], and the values of the pheromone tables after 100 ms (the simulation takes approx. 3 seconds on an 800 MHz Pentium III) is shown in Figure 8.4. Of course, the values in the pheromone tables are dependent on a number of configurable simulation parameters, and they will vary from time to time. Slower or faster convergence can be obtained by choosing other numbers. Further details about the implemented simulator and the parameters are given in the following sections in this chapter.

8.1.4 Ageing of ants and penalty

To ensure that the ants actually find the shortest (or near-shortest) routes between the nodes, they must somehow be encouraged to do so. This is done by progressively reducing the strength of an ant with its age. In this model the strength of an artificial ant is determined by the value of Δp (see the Formula 8.3) using the following formula:

[2]With default parameters, i.e. 155 Mbit/s links with a length of 13 kilometers.

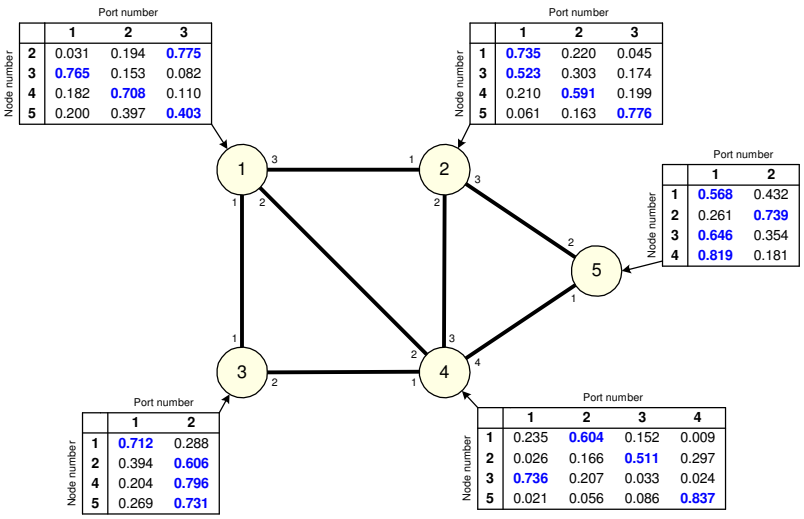


Figure 8.4 The pheromone tables after 100 (simulated) ms.

$$\Delta p = \frac{\Delta p_{max}}{1 + (a/d)} + \Delta p_{min} \tag{8.4}$$

a is the age of the artificial ant specified as a positive integer, and d is the durability of the ant. Δp_{min} and Δp_{max} are *approximately* the minimum value and the maximum value of Δp, respectively[3]. The parameters d, Δp_{min}, and Δp_{max} are configurable in the simulation software. In the current implementation, the values are global — however, they could also be node specific.

8.1.5 Coping with busy links

In non-busy situations, the simulated ants will normally make sure that traffic is routed along (near-)shortest paths. However, if the load of a link ("trail", in the ants' world) gets too high it would usually be desirable to try to use alternative non-shortest paths (if there are any) for future connections until shorter paths once again become less loaded. The artificial ants, however, tend to prefer shorter links over the longer links.

Decreasing the density of artificial ants along busy links can be done by making busy links look longer than they are in reality — i.e. this can be obtained by delaying the artificial ants in a way which increases with the degree of congestion. Since all network nodes in this simulator use output buffers (more about this in

[3]Normally Δp_{max} is much greater than Δp_{min}.

Section 8.3) this effect comes for free, because the delay from traversing loaded buffers will be higher than the buffer delay from traversing non-loaded buffers.

The implemented model supports one additional method of "scaring" the artificial ants away from heavily loaded links. This second method requires that the artificial ant carries with it an extra number — the so-called "penalty". Every time the ant traverses a busy link the penalty is increased by some number, and every time a node updates its pheromone table due to an incoming ant, the penalty is added to the age of the ant. In this way artificial ants travelling over busy links will not contribute so much to the values in pheromone tables. The penalty, Δa, is calculated using the following formula:

$$\Delta a = b_{max} \cdot e^{-b_{gr} \cdot (100-l)} \tag{8.5}$$

l is the load (in %) of the link to which the ant is sent next, b_{max} is the maximum penalty, and b_{gr} (gr: *growth rate*) indicates how fast the penalty grows as a function of the load. Finally, the age of the artificial ant is calculated like this:

$$a = (t - t_c) + \Delta a \tag{8.6}$$

t_c is the creation time of the ant and t is the current time.

8.1.6 Artificial ants with multiple virtual sources

The artificial ants described previously in this section are characterized by their extreme simplicity. However, since an artificial ant is normally travelling over several hops from its source to its destination it could be interesting to investigate whether it could be of any use if the ant could somehow record a few pieces of information along its route. This information could, for instance, be the following:

- The names of the nodes along the route.
- The arrival time at these nodes.
- The input ports the artificial ant used when entering the nodes.

A possible use of the first piece of information in this list could be to make the ant behave as if it had *multiple virtual sources*, even though the ant was only created once. If the actual network technology *requires* that the size of the artificial ant is kept as small as possible, it would not be possible for the ant to carry too much information with it. In this case the artificial ant could just contain the name of *one*

of the nodes along its trail, and using this in addition to the ant's source node, the ant could be seen as if it had *two* virtual sources.

The two types of special ants described here will in the following be referred to as *multi-source ants* and *dual-source ants*, respectively. Two simple examples using these two kinds of ants are given in the following.

Dual source ants

Figure 8.5 illustrates how the network could take advantage of dual-source ants. Once again the route is completely random since this artificial ant is released in a newly initialised network. The ant in this example remembers its source node (as required by all ants) as well as the first node it came to after leaving its source (node 4).

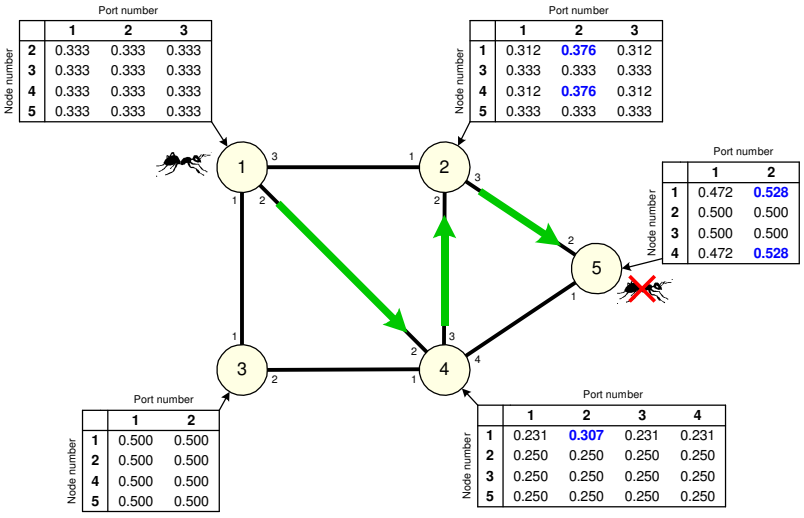


Figure 8.5 The pheromone tables after one dual-source ant has made the journey from node 1 to node 5.

When the ant arrives at node 4 entering port 2, node 4 simply updates its pheromone table as usual, since the ant has only got one source (node 1). After this node 4 adds registers itself as the second (virtual) source of this artificial ant, before the ant is passed forward towards node 2. When the ant arrives at node 2 (and later on at its destination, node 5) the pheromone tables are updated twice, once for each source node.

All-in-all, the effect in this example of having two virtual sources, is that routes (not necessarily optimal routes) have been established from node 5 to node 1 *and*

node 4 (more about routing with ants is explained in Section 8.2). In the example in Figure 8.3, node 5 only knew a route to node 1 and not to node 4. Similarly, node 2 knows a route to node 1 and node 4, whereas node 4 only knows a route to node 1.

Multi source ants

Figure 8.6 shows how it works with multi-source ants.

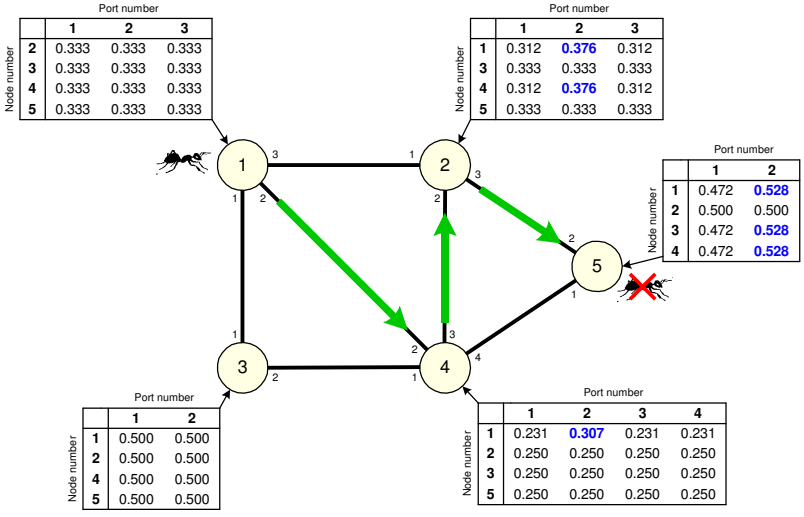


Figure 8.6 The pheromone tables after one multi-source ant has made the journey from node 1 to node 5.

At each intermediate node, the pheromone table is updated for each of the ant's virtual sources (at that time) and the node adds itself to the ant's trail before it is passed forward. This time the single multi-source ant has made it possible for node 5 to find a route to all nodes traversed by the ant (including the source node).

Expectations

The two simple examples with the dual-source and the multi-source ant suggest that the use of these types of ants will lead to faster route discovery compared to the simple ants. Whether the biggest advantage is obtained when dual-source ants remember the first node they visited[4] or when they remember the node they just came from will be shown by the simulations in the remaining part of this chapter.

[4]Like all artificial ants, dual-source ants always remember their *real* source node.

8.1.7 The effect of loops

In the examples so far the artificial ants have been well-behaved. However, what would happen if an artificial ant makes a loop before arriving at its final destination? Figure 8.7 shows an example, where an artificial ant going from node 1 to node 5 makes a loop before arriving at node 5.

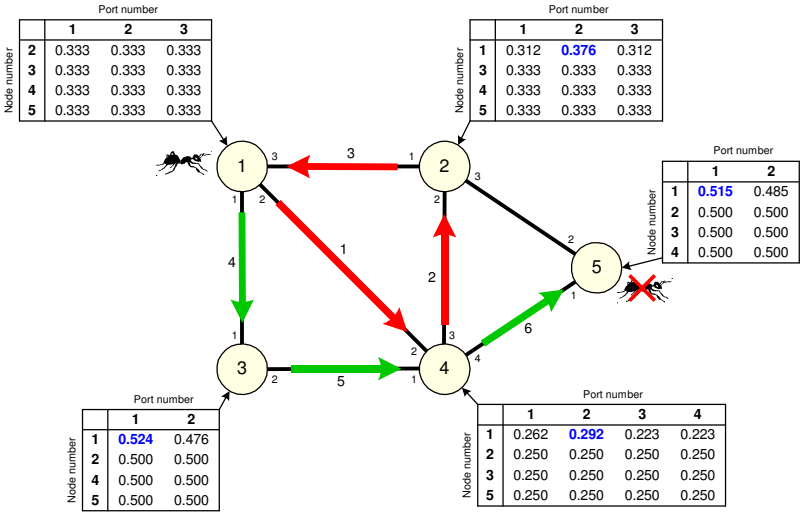


Figure 8.7 The pheromone tables after a (simple) ant has made a loop before arriving at node 5.

As the values in the pheromone tables indicate, no harm is really done compared to the artificial ant choosing a more direct route (see Figure 8.3). For instance, the artificial ant has visited node 4 twice, the first time entering port number 2 and the second time entering port number 1. Nevertheless, node 4 still believes (by quite a margin) that the best route to node 1 goes through port number 2. This is due to the ageing of the ant, so the ant's second visit at node 4 did not have the same impact at the first visit.

8.1.8 Noise

The model described so far works well under (semi-)static conditions. However, since the process of discovering routes can be seen as a positive feedback loop, the artificial ants are not very keen on using alternative routes in case of link failures, congestion, etc. In addition, they lack the ability of finding *new* routes if, for instance, two nodes are suddenly connected by a new link.

To overcome these problems some noise is added to the system. This is done using the same approach as in [42] by simply introducing an extra parameter — the *noise level*. The noise level is the probability that an artificial ant will ignore the values in the pheromone when selecting an output port. For instance, if the noise level is 5%, then there is a 5% probability (at each node) that an ant will choose a random output port, and the probability of using the probabilities in the pheromone table is 95%. If the noise level is 100%, then all artificial ants will move randomly around in the network.

8.1.9 Artificial ant birth rate

The rate by which ants are created (in the nodes) is referred to as the *artificial ant birth rate* or simply the *birth rate*. The birth rate is the probability that an artificial ant is created within any time step. In the simulator (see Section 8.3) the user can choose whether the birth rate is per port (the default setting) or per node. All the results in this chapter are based on the default setting — i.e. the ant birth rate is *per port*.

8.1.10 Balanced versus unbalanced system

For an ant controlled network the terms *in balance* and *out of balance* have the following meaning:

In balance The system is said to be *in balance* when the values in all pheromone tables have reached steady-state. This situation occurs when the artificial ants have moved around in the network for a while under "static" conditions.

Out of balance The system is said to be *out of balance* when sudden changes happen, like — for instance — link/node failures, congestion, changes in the traffic pattern etc. Also, when the initial state of the network (when no ants have been released) is said to be out of balance. When a network is pushed out of balance it will after a while be brought back into balance by the artificial ants.

8.2 Routing

Even though the behaviour of the artificial ants is non-deterministic the ant-based routing mechanism used here is deterministic. Whereas an artificial ant selects an output port with a probability given by the proper entry in the pheromone table, routing is done hop-by-hop by *always* selecting the output port with the *highest* probability.

Routing example

Based on the pheromone tables shown in Figure 8.4 the suggested route from node 5 to node 1 is:

$$\text{node } 5 \rightarrow \text{node } 4 \rightarrow \text{node } 1$$

On the other hand, if node 1 wants the set up a connection to node 5 the following route will be chosen:

$$\text{node } 1 \rightarrow \text{node } 2 \rightarrow \text{node } 5$$

In a network consisting of 5 nodes there are at total of 20 s-d pairs, and all these are easily discovered by the artificial ants after just 100 ms. (Actually, the routes in a network like this are normally found even faster than 100 ms.) The average hop count of the 20 routes suggested by the artificial ants is 1.3.

8.3 Ant simulator implementation details

The ant-controlled network simulator was developed in Java (Sun's JDK version 1.4) in order speed up development time and to obtain platform independence. Even though the speed of Java does not compare to the speed of compiled languages like C or C++, it still gave (more than) adequate speed even when run on an old 800 MHz Pentium III.

During the programming period the focus has been on developing a system that supports circuit-switched as well as packet-switched connections. In this way it is possible to use the simulator to investigate the use of simulated ants together with a lot of technologies like ATM, MPLS, IP, POTS and so on.

8.3.1 Packet types

Four different (fixed-size) packet types have been implemented. The actual size of the packet types depends on the network technology used — so for ATM the size is 53 bytes for each of them. Even though the names of the packet types are borrowed from the terms used in, for instance, ATM and IP, they do not necessarily have any relation to the network protocol being investigated. The packet types implemented in the simulator are the following:

Cells A route between the source and the destination node must be set up *before* cells can be sent through the network. In the actual implementation, cells are *source-routed* — i.e. each cell is at its source node provided with a vector telling the exact route to the destination node. However, the important thing in the simulation is that the route used by cells is fixed within each call.

Datagrams The route of datagrams is decided hop-by-hop — i.e. they are completely self-routing like, for instance, IP packets.

Ants Like datagrams, the route of an ant is decided hop-by-hop. However, whereas the route of datagrams is deterministic, the route of an ant is based on the probabilities in the pheromone tables (i.e. non-deterministic).

Smart Ants Smart ants are the same as (simple) ants, with the exception that they carry more information with them. Dual-source and multi-source ants are special kinds of smart ants.

All packet types are implemented as Java classes, which means that packets are created by instantiating a new object of the desired class. This may sound like a very time-consuming approach, however when packets are moving around in the network only *handles* to the objects are moved, and not the objects themselves. When a packet is not needed anymore, it is simply dereferenced and hereafter dealt with by Java's garbage collector.

8.3.2 The network elements

The following network elements have been implemented in the simulator:

Links Network links are capable of carrying any type of package described above. The length of the network links is specified as the number of (fixed length) packets "floating" on the link at any given moment. For instance, if the link is an optical fibre running at 155 Mbit/s transmitting packets with a size of 53 bytes, a link length of 18 packets corresponds to approximately 10 kilometers of fibre.

Buffers The buffers are needed for coping with temporary congestion in the network nodes. The size of a buffer can be anything from one packet to any number of packets (only limited by the amount on memory in the computer running the simulator). Within a single time-step only *one* packet can be read from the buffer, however any number of packets may be *written* into the buffer within one time-step.

Switches The switches in the simulator are non-blocking with any number of ports (must at least have one port). The switches are capable of switching cells, datagrams (routing), and (smart-)ants. Occasionally, the switches create a new ant with a random destination. Finally, switches serve as sources and destinations for data traffic (datagrams and cells), since no separate traffic sources have been implemented.

Like the packets, all network elements are implemented as Java classes inheriting from the same base class (Equipment.java).

A network node with N ports consists of one $N \times N$ non-blocking switch and N buffers placed at the output of the switch. A simple example showing how the network elements can be connected, is shown in Figure 8.8. The GUI (see Section 8.3.4 or Appendix E) only shows links (bi-directional) and nodes — not the buffers.

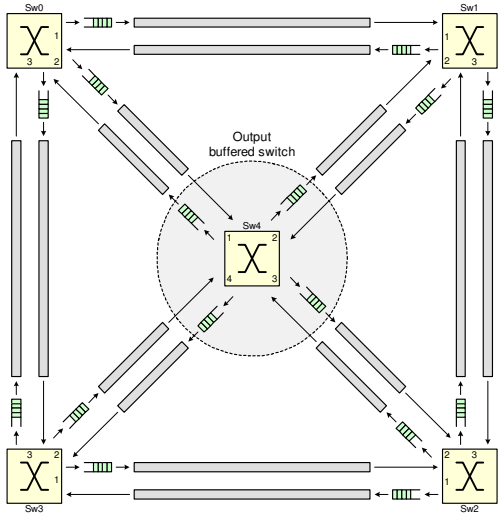


Figure 8.8 The different network elements (links, buffers, and non-blocking switches) supported by the simulator.

The simulator supports network element running at different speeds by specifying a "speed-factor" (must be a positive integer) when creating the Java object for the corresponding network element. Buffers and adjacent links do not need to run at the same speed, but each switch must run at the same speed (or higher) than the fastest link connected to the switch — otherwise packets will be lost.

Connectors

Connecting the network elements to one another is done using a number of simple Java classes inheriting from a common class (Connector.java). Objects instantiated from these classes are referred to as connectors. Basically, what the connectors do is simply passing packets from on type of equipment to another, every time the method `clock()` is called.

8.3.3 Simulation time-step

Figure 8.9 shows the method calls involved in one simulation time-step.

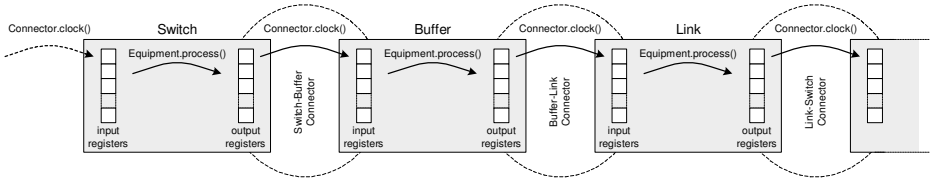


Figure 8.9 One time-step in the simulator.

All network elements implement a method called `process()`. Invoking this method causes the network element to process the data in the input registers and place the result in the output registers. For instance, the link model simply copies the input registers to an internal queue, and moves one element from the queue to the output registers.

So if all network elements are to run at the same speed, one step in the simulator corresponds to first invoking the method `process()` for all network elements, and then invoking `clock()` for all connectors in the network. This is repeated again and again.

However, if some pieces of equipment run faster than the base speed (i.e. the speed-factor is greater than one) `process()` and `clock()` must be called several times. For instance, if two switches (sw1 and sw2) are connected through a link running at triple speed the following must be done to move forward by one time-step:

1. Call `process()` for all network elements.
2. Call `clock()` for all connectors.
3. Call `process()` for sw1, sw2, the buffers of the two switches, and the link between them.

4. Call `clock()` for all connectors between sw1 and sw2.

5. Repeat 3.

6. Repeat 4.

8.3.4 Graphical user interface

A graphical user interface has been build to quickly get some qualitative results from simulating an ant-controlled network. Through the GUI, the user can observe the effects of:

- Destroying (and repairing) links or network nodes.
- Adding (or removing) circuit-switched or packet-switched connections.
- Changing the basic parameters (ant birth rate, feedback strength, and much more).

More about the GUI and how to use it can be found in Appendix E

8.3.5 Parameters, options, and simulation modes

The simulator supports quite a few parameters and options through which the behaviour of the artificial ants can be altered. Figure 8.10 shows two screenshots of the simulator's options window. In Figure 8.10a simple ants are used, whereas Figure 8.10b shows a situation where multi-source ants are used and failing links are dropped almost immediately by altering the values in the pheromone tables of the nodes adjacent to a failing link. Furthermore, using the options shown in Figure 8.10b will case the immediate killing of looping ants (once they are discovered). The values of the parameters shown in Figure 8.10 are the default values used in most simulations presented in this chapter.

The model described in Section 8.1 is dependent on the parameters shown here. However the GUI uses more descriptive names instead of the mathematical symbols used in Section 8.1. The mapping between the two representations is shown in Table 8.2.

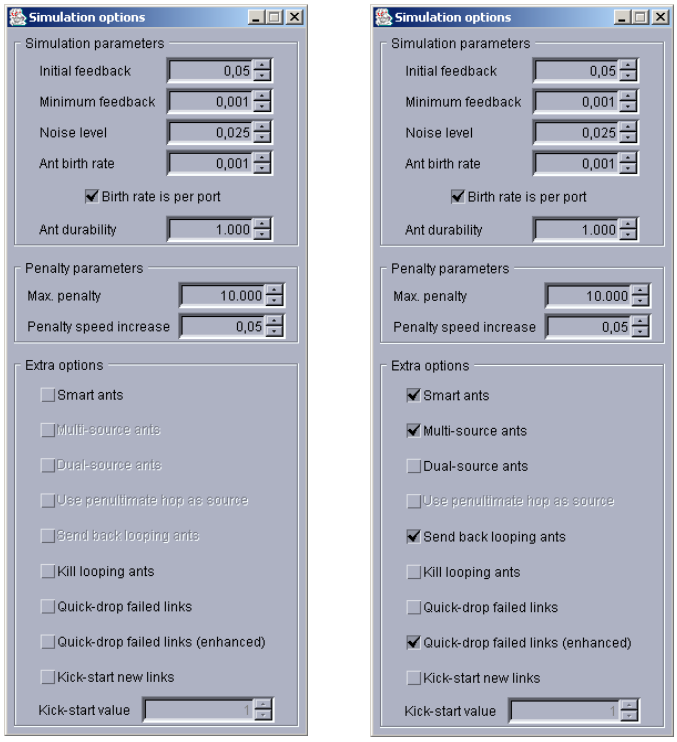


Figure 8.10 Screenshot of the simulator options window.

Table 8.2 Mapping between symbols and GUI names.

Symbol	GUI name
Δp_{max}	Initial feedback
Δp_{min}	Minimum feedback
(no symbol)	Noise level
(no symbol)	Ant birth rate
d	Durability
b_{max}	Max. penalty
b_{gr}	Penalty growth rate

The effect of selecting the different options shown in Figure 8.10 is as follows:

Smart ants causes the switches to generate smart ants instead of "normal" artificial ants. Smart ants remember the nodes they have visited and the time they arrived at each node.

Multi-source ants are artificial ants being treated as if they have multiple virtual sources.

Dual-source ants are a special case of multi-source ants (ants with two virtual sources). These ants remember their source node as well as the *first* node they visited. Dual-source ants and multi-source ants can not be selected at the same time.

Use penultimate hop as source causes (when combined with dual-source ants) the artificial ants to remember their source node as well as the node they just came from (the pen-ultimate node).

Send back looping ants causes looping ants to be sent back to their source where they will be terminated. This option requires smart ants since the trail must be remembered in order to go back using the same route — however, dual-source ants cannot be combined with this option, since they only remember two virtual sources, and not their *entire* route. When ants are sent back to their source, they try to *undo* the changes in the pheromone tables they caused on their trip away from the source.

Kill looping ants causes looping ants to be killed immediately once they are discovered. I.e. these ants never make it to their destination. In practical implementations, loops may be detected in two ways. If the ant remembers all the nodes it has visited (smart ants), loop detection is trivial. However, even without smart ants loops may be detected if all nodes keep a history log of all ants it has seen within a reasonable amount of time — this loop detection mechanism requires that the artificial ants are provided with a unique identification. If an ant is killed by accident once in while, because it is mistaken for another ant, in will not influence the overall performance of the system.

Quick-drop failed links speeds up the process of finding alternative routes through the network if a link fails (or if several links fail). This is done by immediately setting the probability of the ports connected to the failed link to zero (in the pheromone tables) — the old value is split *evenly* among the other ports to ensure that all probabilities still sum up to 1.

Quick-drop failed links (enhanced) does almost the same, except that the old value (corresponding to the failing port) is *not* split evenly among the other ports. Instead the other ports are given a portion proportional to their current value. Whether this minor change will have any noticeable effect will be shown by the simulations in Section 8.6.

Kick-start new links is an option which can be useful to speed up the discovery of new links (or to re-populate links that were emptied due to a temporary link failure). However, this option is not used in this chapter.

A complete list of the possible combinations of options (excluding "quick-drop failed links", "quick-drop failed links (enhanced)", and "kick-start new links") is given in Table 8.3.

Table 8.3 Mapping between mode numbers and simulation settings.

Mode	Simulation settings
1.	Simple ants (no options selected)
2.	Simple ants + Kill looping ants
3.	Smart ants + Send back looping ants
4.	Smart ants + Multi-source ants
5.	Smart ants + Multi-source ants + Send back looping ants
6.	Smart ants + Multi-source ants + Kill looping ants
7.	Smart ants + Dual-source ants
8.	Smart ants + Dual-source ants + Kill looping ants
9.	Smart ants + Dual-source ants + Use penultimate hop as source
10.	Smart ants + Dual-source ants + Use penultimate hop as source + Kill looping ants
11.	Smart ants
12.	Smart ants + Kill looping ants

The two last combinations shown in Table 8.3 are not interesting, since they are in effect identical to the two first combinations. The combinations (11) and (12) do indeed cause the switches to produce smart ants, however the extra information carried around by these ants is not used at all.

8.4 Simulation parameters

This section examines the influence of the values of some of the simulation parameters. The focus here is mainly on finding out when *stability* issues may occur, in order to ensure that the ant controlled system behaves in a predictive manner. The following parameters will be examined:

- Initial feedback, Δp_{max}
- Noise level
- Ant birth rate

The network used for these examinations is the mesh network shown in Figure 8.14 (see Section 8.5). Also, it is assumed, that the network is using the ATM protocol and an artificial ant is simply a special ATM cell. All links are running at 155 Mbit/s and the length of the links are approximately 13 kilometers (see also Section 8.5). Since the number of network nodes is 28, the total number of s-d pairs is 756. Finally, simple ants (mode 1) have been used for the simulations in this section, and no data traffic has been added to the network.

The influence of the value of the initial feedback is shown in Figure 8.11. The figure shows the number of discovered routes between the s-d pairs as a function of time and the value of the initial feedback parameter. Only 80 ms are actually shown here, but the picture remains the same even though the simulation is continued for several seconds.

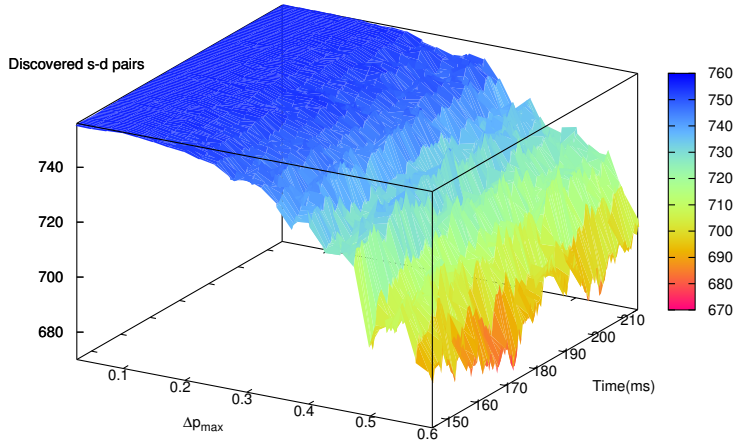


Figure 8.11 The number of routes between s-d pairs as a function of the time and the initial feedback, Δp_{max}.

From the Figure 8.11 it is clear that the value of Δp_{max} should remain below 0.10, otherwise the system will never be stable. If Δp_{max} gets larger than 0.10, each individual artificial ant will have too much influence, and since individual ants behave in a non-deterministic manner, this non-deterministic behaviour is "transferred" to the whole system if the impact of a single ant gets too high.

Figure 8.12 shows the number of discovered routes between s-d pairs as a function of time and the noise level. As expected, the noise level must be kept reasonably low, otherwise the artificial ants are not capable of finding routes between all s-d pairs. Noise levels up to 0.10 (10%) are tolerable, however the noise level should

not be too low either, since that would have a significant impact on the dynamics of the system (see Section 8.1.8).

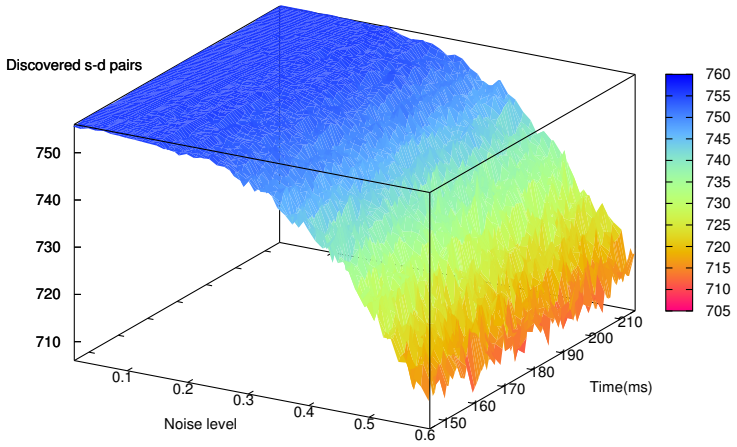


Figure 8.12 The number of routes between s-d pairs as a function of the time and the noise level.

Finally, Figure 8.13 illustrates the impact of the ant birth rate. As expected a high ant birth rate means that the routes between the s-d pairs are discovered faster than if the ant birth rate was lower. However, it is also important to keep the ant birth rate within reasonable limits, otherwise the artificial ants would "steal" too much bandwidth from data traffic (see Section 8.5).

8.4.1 Parameters used for simulations

Based on the results presented in Section 8.4 as well as experience, the values listed in Table 8.4 were used for the simulations in the following part of this chapter. The ant durability has been set to 1,000 in all simulations, however the exact value of that parameter is not critical.

8.5 Network topology

In this section it is examined how artificial ants behave when used with different network topologies. The intention of this thorough examination is to find out whether artificial ants can be used with any kind of topology, or whether they are

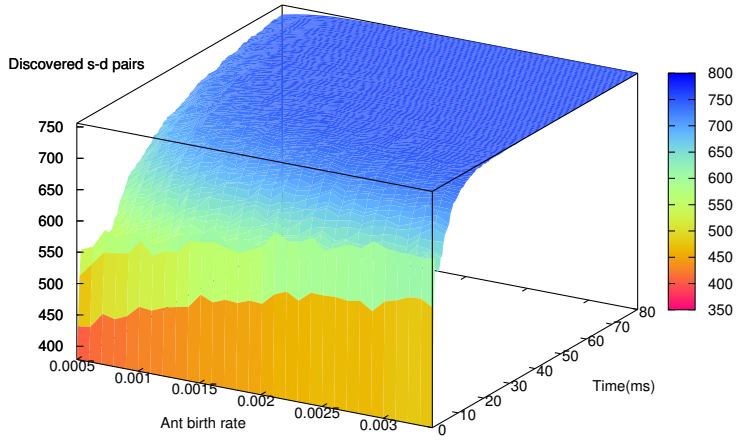


Figure 8.13 The number of routes between s-d pairs as a function of the time and the ant birth rate.

Table 8.4 Selected simulation parameters.

Parameter	Value
Initial feedback	0.05
Minimum feedback	0.0005
Noise level	0.025
Ant birth rate	0.001
Ant durability	1,000

only useful in special cases. Three networks using different topologies (mesh, tree, and ring) have been examined thoroughly in terms of the following:

- How fast are the routes between nodes (s-d pairs) discovered?
- How does the (average) hop-count change with time?
- What is the link load induced by the artificial ants?

These things are examined for the mode numbers 1–10 (see Table 8.3) and the default parameters (see Figure 8.10) and without any data traffic (however, it turns out that adding traffic does not make any noticeable difference). Not all results from the simulations are presented in this section. The results that have been left out can be found in Appendix D.

In order to get quantitative result, it is assumed that the networks examined in this section are running ATM, and that all link speeds are 155 Mbit/s. The buffers in the nodes (ATM switches) may contain a maximum of 1000 cells, and all links have a length of approximately 13 kilometers[5]. An artificial ant is simply an ATM cell[6] with a special tag (e.g. special VPI/VCI values). Furthermore, all simulations have been repeated 100 times, and the presented results are the average of the 100 simulation results.

The choice of using ATM is completely arbitrary. Actually, the networks could be running virtually *any* protocol. In that case the packet sizes just have to be adjusted to a typical size.

8.5.1 Mesh network

The first network being examined is the mesh network shown in Figure 8.14. This network consists of 28 nodes (ATM switches) and 56 links. The number of possible routes between source and destination nodes is $28 \times 27 = 756$ (and *not* $\frac{1}{2} \times 28 \times 27$). The reason for this is that the artificial ants create trails in only one direction — the opposite direction of their motion. Due to this it is quite common (in an unbalanced system) that a route exists in *one* direction between two nodes, but not in the other direction — and, of course, there has to be a route in both directions before two nodes can be considered as connected.

[5]This odd number occurs because it is assumed in *this* section that a maximum of 25 ATM cell can be "floating" on a link at any given moment. At 155 Mbit/s it takes 2.7 μs to process an ATM cell, so processing 25 ATM cells takes 68 μs. The speed of light in a fibre is approximately $\frac{2}{3} \cdot c$, and in 68 μs the light can travel **13.6 kilometers**.

[6]A multi-source ant does not necessarily fit into a single ATM cell, if its trail gets too long. However, the result presented here still assume, that multi-source ants fit into one cell.

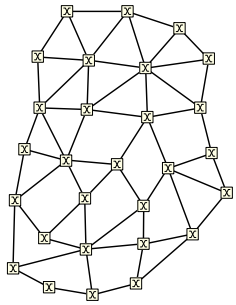


Figure 8.14 Mesh network used for simulations.

Figure 8.15 shows the number of discovered routes between s-d pairs as a function of time. In order to get nice and smooth curves that can easily be compared to each other, the results presented here are averaged over 100 runs. These 100 runs have been made for each of the modes 1–10 described in Table 8.3, however, only the results of 5 of the modes are included in this chapter. The results of the remaining 5 modes can be found in Figure D.1 in Appendix D.

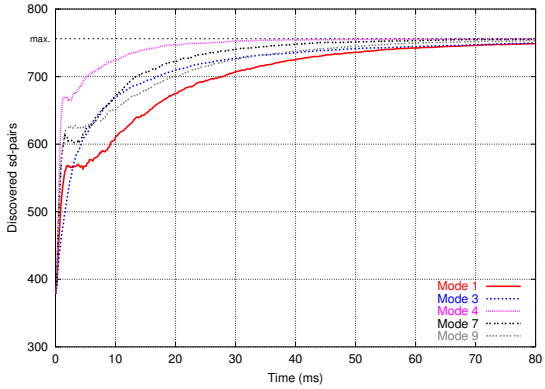


Figure 8.15 Discovered s-d pairs as a function of time using the mesh network shown in Figure 8.14.

From Figure 8.15 it is clear that the fastest way of discovering the routes between the s-d pairs is using the multi-source ants without killing or sending back looping ants (mode 4) whereas the simple ants (mode 1) are considerably slower.

Figure 8.16 shows the average hop-count of the discovered routes as a function of time. It is somewhat surprising to see that in the beginning the average hop-count of the discovered routes is much higher than the optimal average hop-count (when all routes have been discovered), because the first nodes to find one another will

typically be neighbour nodes. However, it only takes in the order of 10 ms before the average hop-count has moved considerably closer to the optimum, except for mode 3 where the looping ants are sent back to their source.

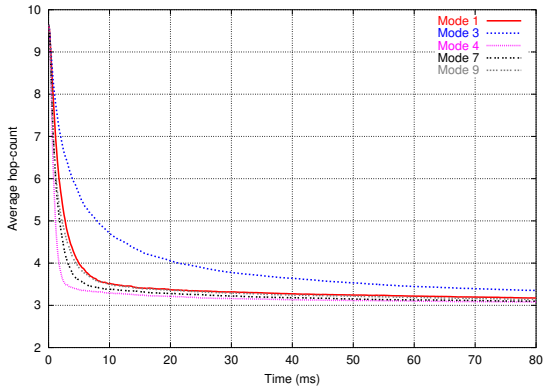


Figure 8.16 Average hop-count (of the discovered routes) as a function of time using the mesh network shown in Figure 8.14.

The Figures 8.17 and 8.18 show the maximum load and the average load induced by the ants, respectively. When using mode 3 (where looping ants are sent home) the average and the maximum load is almost constant. In the other situations, however, the ant-induced load rises in the beginning and as more and more routes are established the load decreases slowly to a steady level. The increased load while the network is out of balance must be considered to be caused by looping ants. This becomes even clearer when looking at Figure D.2 in Appendix D. In all situations whereas looping ants are either killed or sent back to their source the ant-induced load is almost constant, which it is not in the other cases.

8.5.2 Tree network

Actually, tree networks are not the most interesting topology to examine due to their extreme simplicity. No alternate routes exist between nodes, so traffic between any two nodes can not be split among several routes. So the intention of this examination is mainly to find out whether artificial ants *can* be used with tree networks.

The tree network shown in Figure 8.19 consisting of 31 nodes and 30 links is used for the simulations in this section. The results not presented here are shown in the Figures D.3 and D.4 in Appendix D.

Figure 8.20 shows the numbers of discovered routes as a function of time. Once again, the multi-source ants perform best, and the absolutely worst results are ob-

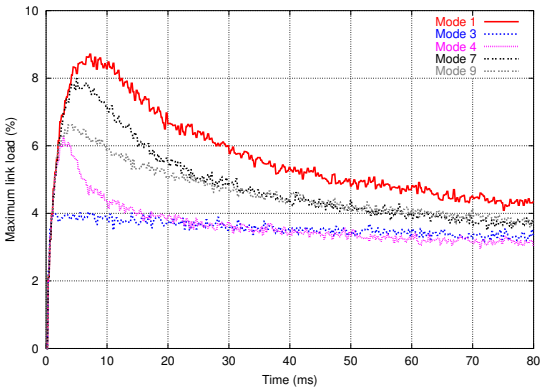


Figure 8.17 Maximum link load in percent induced by the artificial ants as a function of time using the mesh network shown in Figure 8.14.

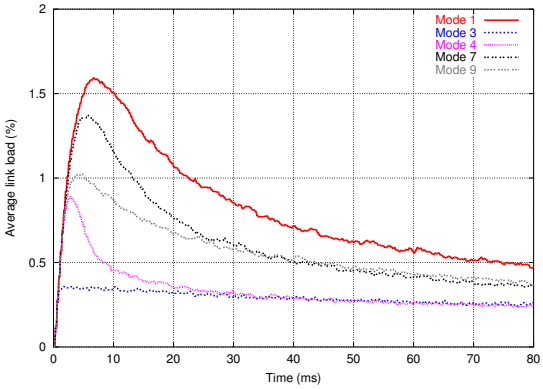


Figure 8.18 Average link load in percent induced by the artificial ants as a function of time using the mesh network shown in Figure 8.14. The average load is calculated by dividing the sum of all link loads with the number of links.

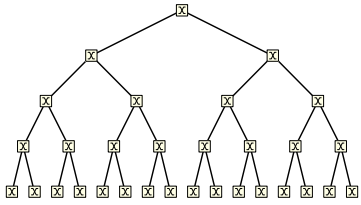


Figure 8.19 Tree network used for simulations.

tained when looping ants are sent back to their source.

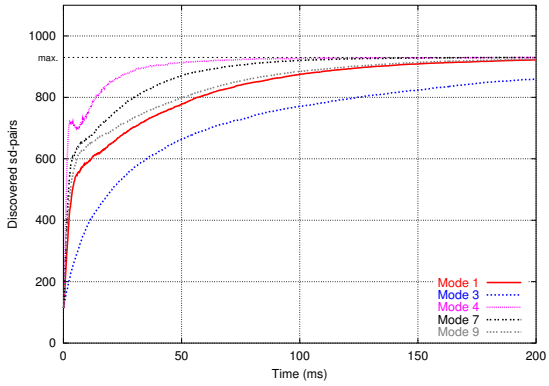


Figure 8.20 Discovered s-d pairs as a function of time using the tree network shown in Figure 8.19.

The average hop-count of the discovered routes are shown in Figure 8.21. This figure is quite different from Figure 8.16, and the reason for this is that in mesh networks the ants tend to find non-optimal routes very quickly, and then gradually, the routes get closer to the optimum. In tree networks, however, non-optimal routes do not exist — all routes are optimal.

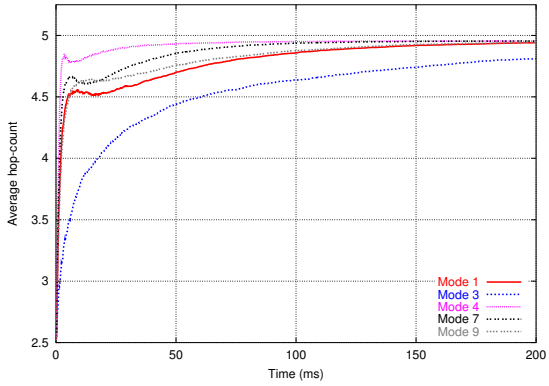


Figure 8.21 Average hop-count (of the discovered routes) as a function of time using the tree network shown in Figure 8.19.

The maximum and the average loads induced by the artificial ants are shown in the Figures 8.22 and 8.23, respectively. Once again the load is near-constant in those situations where looping ants are either killed or sent back. Furthermore, the ant induced load is considerably higher than for mesh networks. This is not surprising,

though, since all ants travelling from one main branch of the tree to another needs to go through the root node.

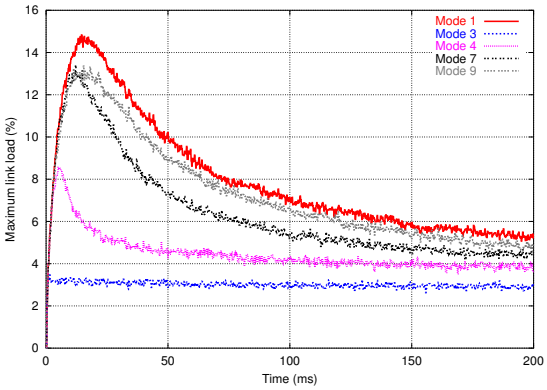


Figure 8.22 Maximum link load in percent induced by the artificial ants as a function of time using the tree network shown in Figure 8.19.

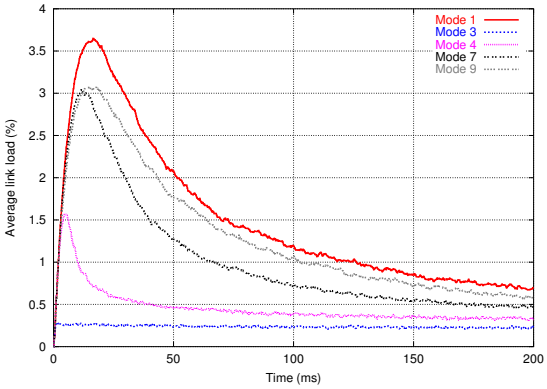


Figure 8.23 Average link load in percent induced by the artificial ants as a function of time using the tree network shown in Figure 8.19.

8.5.3 Ring network

Ring networks have been a popular topology when used with FDDI, SDH, optical networks, etc. Presumably, the main reason for the popularity of ring networks has been their simplicity, both in terms of routing and fault management. For instance, in SDH/SONET BLSRs (bi-directional line-switched rings) traffic can be switched from working fibres to protection fibres within 50 ms [95].

The ring network used for testing is shown in Figure 8.24. In consists of 28 nodes (ATM switches) — i.e. with a link length of approximately 13 kilometers, the total length of the ring is almost 400 kilometers.

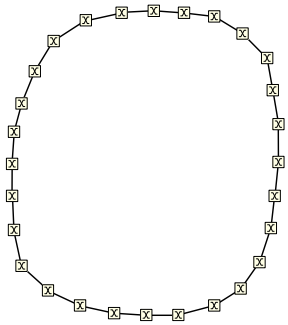


Figure 8.24 Ring network used for simulations.

Figure 8.24 shows how quickly the artificial ants discover the possible routes and Figure 8.26 shows the average hop-count of the discovered routes (more results are shown in Figure D.5). As expected the multi-source ants perform best, whereas the worst results are obtained when looping ants are sent back to their source.

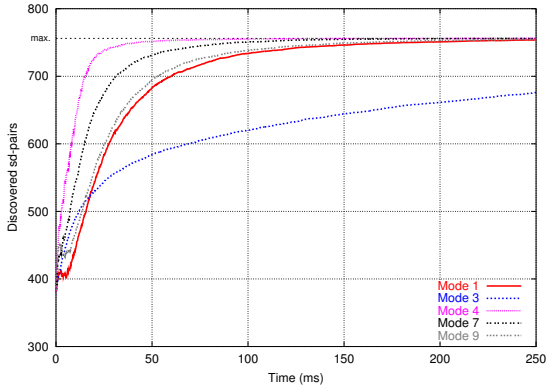


Figure 8.25 Discovered s-d pairs as a function of time using the ring network shown in Figure 8.24.

The Figures 8.27 and 8.28 show the maximum and average load induced by the ants. As for tree networks, the load peaks at a considerably higher values than the ant induced load in mesh network. The result that have been left out here can be seen in Figure D.6.

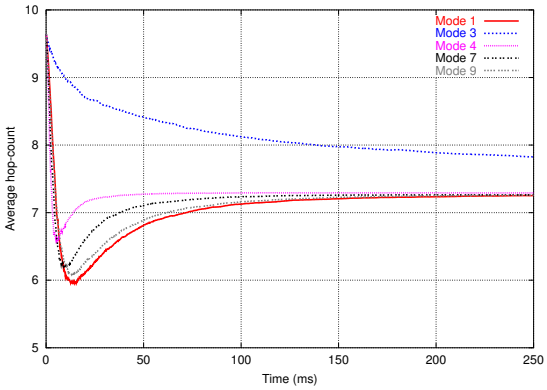


Figure 8.26 Average hop-count (of the discovered routes) as a function of time using the ring network shown in Figure 8.24.

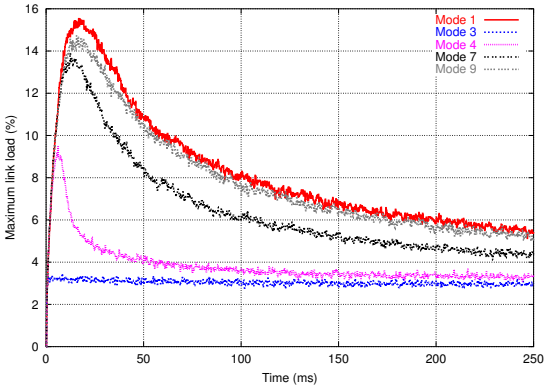


Figure 8.27 Maximum link load in percent (from the artificial ants) as a function of time using the ring network shown in Figure 8.24.

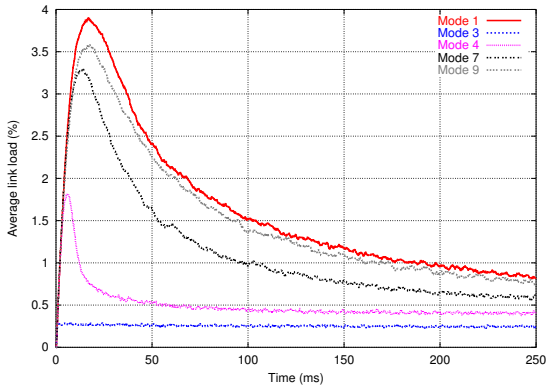


Figure 8.28 Average link load in percent (from the artificial ants) as a function of time using the ring network shown in Figure 8.24.

8.5.4 Comparison of mesh, tree, and ring topologies

The result from the topology examinations are compared in the Figures 8.29, 8.30 and 8.31. They illustrate the time is takes before 90%, 99%, and 100% of the routes are discovered — in this case the results from *all* the 10 possible modes are shown.

All modes, where looping ants are either killed or sent back to their source (i.e. the modes 2, 3, 5, 6, 8, and 10), lead to bad results for the ring topology and to some degree for the tree topology. Moreover, the modes 4 and 7 (using multi-source and dual-source ants) give the best results in all cases, with the multi-source ants having a slight edge over the dual-source ants.

On the other hand, the dual-source ants using the penultimate hop as the second virtual source (mode 9), does *not* lead to significant improvements over the simple ants (mode 1). The reason for this seems to be that mode 9 makes it easier for the artificial ants to establish routes between *neighbouring* nodes, which is not a big problem for the simple ants.

The results shown in Figure 8.31 are actually worst-case results. The reason for this is that out of the 100 runs, a single "unlucky" run lead to a bad *average* result. For instance, if all 756 routes are discovered after 200 ms for 99 of the 100 runs, and "only" 754 routes are found for the after 200 ms for the last run, then — on the average — "only" 99.997% of the route have been discovered. This effect is not so evident for the 90% and 99% results, so their uncertainty is much lower.

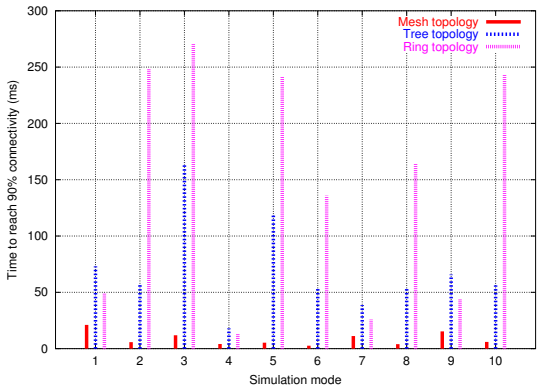


Figure 8.29 Time before 90% of the all possible routes are established.

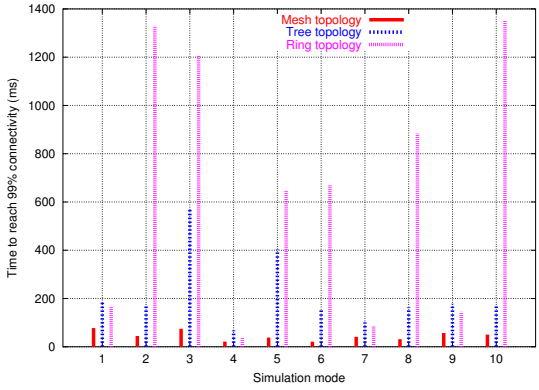


Figure 8.30 Time before 99% of the all possible routes are established.

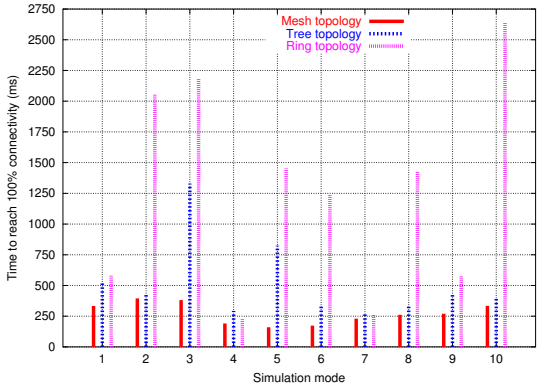


Figure 8.31 Time before *all* possible routes are established.

8.5.5 Dense versus sparse networks

In graph theory graphs are often categorized into the following two categories [4, 20][7]:

- If E is close to V^2 the graph is *dense*. In essence, this means there is an edge connecting almost any two given vertices.
- If E is much less than V^2 the graph is *sparse*.

In communication networks, however, talking about *dense networks* does not make much sense, because it goes against the very idea of building networks in the first place. If it was possible to interconnect all sites in the world directly with optical fibres, a global communication network would not be needed. But in that case the required number of optical fibres would be $\frac{1}{2} \times N \times (N-1)$, where N is the number of sites (not to mention the *length* of the fibres).

To some degree, the Sections 8.5.1, 8.5.2, and 8.5.3 have already given the answer to what to expect about the behaviour of artificial ants, when used with networks with varying densities. Ring networks and a tree networks are examples of extremely sparse networks, so based on the previous results, it is expected that artificial ants are least efficient with sparse networks.

The simulations here are based on the network shown in Figure 8.32a, which is almost identical to the mesh network shown in Figure 8.14 — only with a few extra links — so this network contains 28 switches and 66 links. In order to turn 28 switches into a fully connected network, a minimum of 27 links are required. This means that the test network used here contains $66-27 = 39$ links more than is *absolutely* required to obtain a full connectivity.

In the simulator links and switches can be disabled and enabled at any time. Furthermore, an algorithm has been implemented allowing a number of links to be disabled and *still* maintaining full connectivity. Using this algorithm, 50 runs have been made with 0 to 39 disabled links (i.e. a total of $40 \times 50 = 2000$ runs). In effect, this approach is equivalent to doing runs with networks of varying density.

The Figures 8.32b and 8.32c show the network with 10 and 39 disabled links, respectively. When 39 links are disabled, the resulting network is actually a tree network and from the previous results, the artificial ants are expected to react slower with tree networks.

Figure 8.33 is the outcome from those 2000 simulations, showing the number of discovered routes between all s-d pairs as a function of the elapsed time (in mil-

[7] E is the number of edges and V is the number of vertices in the graph.

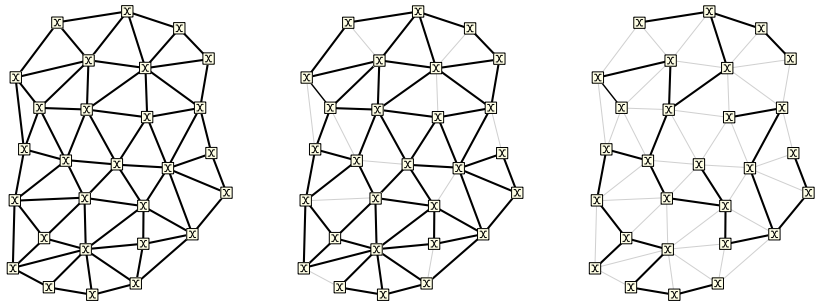


Figure 8.32 Mesh network used to test how ants cope with sparse and dense networks. (a) shows the network with all links enabled, whereas 10 links have been disabled in (b). In (c) 39 links have been disabled so that the resulting network is actually a tree network.

liseconds) and the number of (randomly chosen) *disabled links*. Simulation mode number 7 (dual-source ants) has been used for this simulation, however, all modes lead to qualitatively similar results.

From this figure it is clear that the more sparse a network becomes (i.e. the more links that are disabled), the longer time it takes for the ants to establish the routes — i.e. to get into balance. This does not mean that artificial ants can not be used for sparse networks, it just takes more time to reach balance. For the case with 39 disabled links all routes between the possible s-d pairs will be discovered after 3–5 seconds (not shown in the figure).

8.6 Fault management

Fault management — which is an important aspect of resource management — actually comes for free in ant controlled networks. If a link fails, the ants will stop arriving from that link which, in turn, will cause the pheromone strength of that link to decrease over time. More precisely, if a link connected to port j of a given node fails, all values in column j of the pheromone table — i.e. $p(1, j)$, $p(2, j)$, …, $p(n, j)$ — will slowly approach zero (see Table 8.1). This means that after a while the routing mechanism (see Section 8.2) will stop using a failed link.

It is obvious that routes along failed links should be avoided as quickly as possible. However, this is not the only aspect of fault management. After failed links have been repaired they should be used again as quickly as possible. The speed by which repaired links return to their normal activity is not as critical as recovering

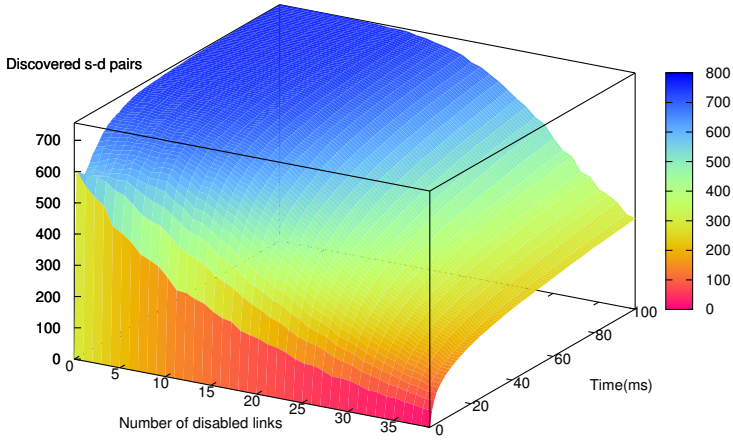


Figure 8.33 Time to reach steady state with 0–39 disabled links (mode 7).

from link failures. It is, nevertheless, important that they will become used at *some* time — and the faster the better. The noise level of the system (see Section 8.1.8) is important for determining how quickly repaired links return to their original level of usage.

The first part of this section investigates how quickly the artificial ants can recover from links failures, whereas the last part investigates what happens when links are repaired.

Once again, the mesh network shown in Figure 8.32 is used for these investigations, and the length and the speed of the links are as described in Section 8.5 (i.e. 155 Mbit/s links with a length of 13.6 kilometers running ATM).

8.6.1 Recovery from link failures

If a link fails, all routes between any two nodes using that link will be disrupted. Obviously, if several links fail the number of disrupted routes will be even higher. In this section it is investigated how quickly the disrupted s-d pairs will find alternative routes.

Of course, even though there is a valid route between two nodes, there is no guarantee that there is actually enough spare capacity to support a connection with a certain bandwidth. However, the existence of a route is a necessary prerequisite for making it possible to set up a connection in the first place.

When using the network shown in Figure 8.32 it is possible to maintain a fully connected network even though as many as 39 (carefully selected) links are failing simultaneously. The simulations are made as follows:

1. At the time $t = 0$ms the simulation is started.
2. In order to get faster into balance, the ant birth rate is heavily increased during the first 100 ms of the simulation. The result of this is that all routes between the s-d pair are found (and their lengths are optimal) already after 80–100 ms.
3. At the time $t = 200$ms a number of links are suddenly disabled.
4. After this, the simulation is continued for a while, and the number of discovered routes is monitored.

All the simulations in this section are done without background traffic. However, tests have been made with various traffic patterns, and they have shown a similar behaviour.

Moreover, the fault management results shown in this chapter are all simulated using mode 7 (dual-source ants). However, these simulations have been carried out for the modes 1, 4, 7, and 9. All results can be seen in the Section D.3 in Appendix D.

No detection mechanism in the nodes

As already mentioned, fault recovery comes for free due to the behaviour of the artificial ants. The results shown in Figure 8.34 show the number of discovered routes between s-d pairs as a function of time and the number of failing links (averaged over 20 runs). In this scenario, the switches do not monitor the status of the links — i.e. they do not detect that anything is wrong. The fault management is entirely the responsibility of the artificial ants.

Figure 8.34 shows that if only one link fails, approximately 25 out of a total of 756 routes are lost, but approximately 150 ms later they have been rerouted to alternative paths. On the other hand, if 39 links suddenly fail 650 s-d pairs are disconnected, and it takes several seconds for the artificial ants to find alternative routes. All in all, this scenario gives fairly good results if few links fail, but rather bad results if many links fail.

The failure of a *switch* corresponds to a number of simultaneous link failures — i.e. a situation with many link failures is not completely unlikely. So, fast recovery from multiple link failures is certainly a desirable feature.

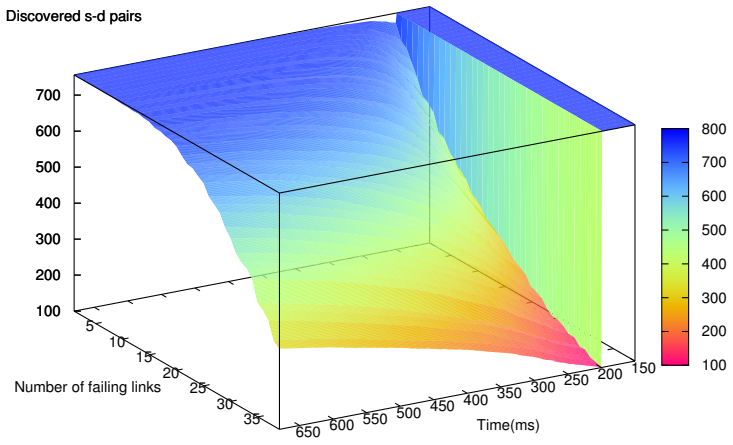


Figure 8.34 The number of routes between s-d pairs as a funtion of time and the number of failing links using **mode 7**.

Failing links are detected by the nodes (version 1)

A simple improvement could be to make the switches react immediately when they discover that one or more of its links have failed. Most (if not all) switches in real networks monitor the link states anyhow, so they might as well use that information to assist the artificial ants in doing their job.

So, if a switch detects that one of its links is not working anymore, it *immediately* clears (i.e. sets to zero) all the values in the pheromone table corresponding to the port connected to the failing link. The old values are then distributed evenly to the other ports in the pheromone table to make sure that each row still adds up to 1.

Figure 8.35 illustrates the effect of this simple addition to the nodes' fault management mechanisms. The effect of a few (say, 1–8) simultaneous link failures is barely visible, and even with 39 failing links more than 700 out of 756 routes between s-d pairs have been found after just 500 ms.

Failing links are detected by the nodes (version 2)

The results become even better by making a tiny modification to the previously described approach. Instead of distributing the values of the cleared entries evenly to the other entries, the share depends on the previous value of each of the other entries. In the simulator, this is implemented as follows:

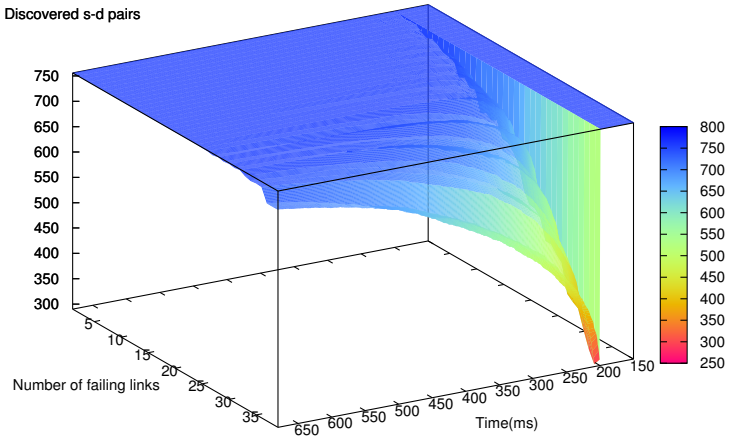


Figure 8.35 The number of routes between s-d pairs as a funtion of time and the number of failing links using **mode 7** and a mechanism to quickly drop broken links.

A switch detects, that the link connected to port number v does not work anymore. Immediately, all values in column v of the switch's pheromone table are cleared (see Table 8.1).
After this, each row in the pheromone table is *normalized* — i.e. the values are adjusted so that the sum of each row is 1. This is done by first calculating the sum of the values in each row, i:

$$s(i) = \sum_{j=1}^{m} p(i,j)$$

All values in each row, i, are then divided by the sum of the values for that row, $s(i)$:

$$p(i,j) = \frac{p_{old}(i,j)}{s(i)}, \ \forall \ j$$

This is repeated for all rows, and finally the sum of each row is once again 1.

Figure 8.36 shows how quickly the routes are restored when the nodes use this enhanced fault detection (and recovery) mechanism. In this case all routes are restored after approximately 500 ms, even in the case where 39 links are failing at the same time.

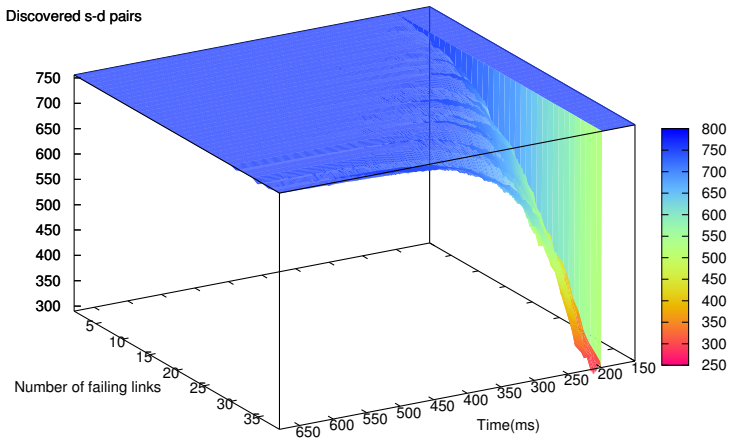


Figure 8.36 The number of routes between s-d pairs as a funtion of time and the number of failing links using **mode 7** and an enhanced mechanism to quickly drop broken links.

8.6.2 Adding/repairing links

This section studies the effects of repairing links, which actually corresponds to adding new links to the network. Two things are interesting in the case that links are repaired:

- How many (if any) routes are temporarily lost while the system gets into balance?
- How quickly will the repaired (or new) links be used?

It might seem strange, that adding new links could lead to *lost* routes. However, if many links are added at the same time, the system can be considered to be seriously out of balance, and this may lead to unexpected behaviour.

In the following simulations, the initial state is a system in balance with a number (1–39) of disabled links — i.e. all routes are discovered and the average hop-count is very close to the optimum. The average optimum average hop-count will naturally depend on the number of disabled links. The more disabled links, the higher the optimum average hop-count. At the time $t = 0$ms all links are simultaneously enabled.

The results presented here are based on mode 7. The results when using the modes 1, 4, 5, 8, 9, and 10 are shown in Section D.4 in Appendix D.

Figure 8.37 shows the number of lost routes as a function of the number of recovering links and the time after the recovery takes place. After approximately 100–200 ms a few links are lost — but only in the situation where 15–20 or more links are enabled at the same time. After 1500 ms, virtually all routes are found again. Enabling 1–15 links at the same time does not lead to any instability issues.

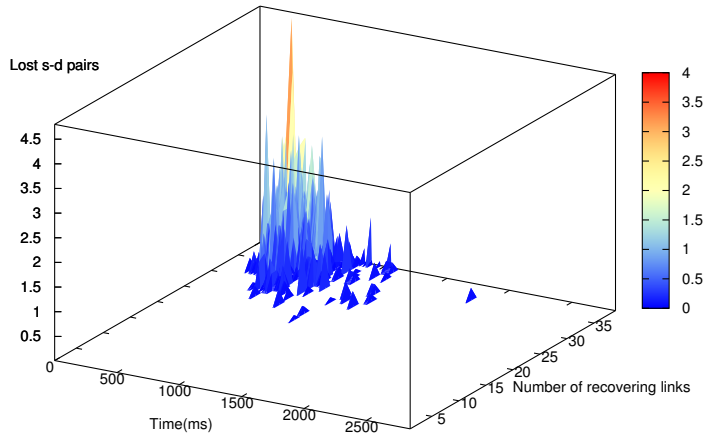


Figure 8.37 Number of lost routes as a function of time and the number of links being enabled (**mode 7**).

Figure 8.38 gives an illustration of how quickly the recovering links will be used again. Almost nothing happens to the average hop-count for the first 200 ms or so, but then the average hop-count starts decreasing. The decreasing hop-count actually proves that the recovered links are used. After approximately 3 seconds all the recovered links have returned to their "normal" state.

8.6.3 Summary

Even though the artificial ants provide really good fault management by themselves, the results in this section have shown that great improvements can be made by combining link monitoring with ant principles. Even though the simulated scenarios with up to 39 failing (or recovering) links exceeds anything that might be expected in real networks, the results in this section have shown that the artificial ants are capable of dealing with these worst-case situations.

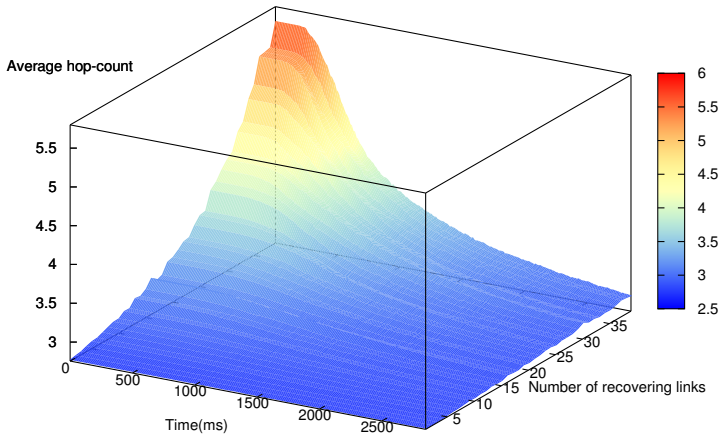


Figure 8.38 Average hop-count of the routes after link recovery (**mode 7**).

Studies of routing in the Internet have shown that it may take several minutes for the routers to reach a consistent view of the network topology after single faults [96]. With this in mind, it is clear that fault recovery mechanisms of networks controlled by artificial ants are very fast indeed.

8.7 Load balancing

Load balancing is about distributing the load as much as possible on equal or near-equal cost links. In this way the occurrence of network congestions can be postponed.

Load balancing in ant controlled systems comes from the fact that the pheromone trails deposited by ants traversing highly loaded links will not be as strong as the pheromone trails deposited by ants traversing links with a lower load. This section will give the results from a few examples illustrating the principles of load balancing using artificial ants. This is followed by some comments on the pros and cons of load balancing using the types of ants implemented in this project.

8.7.1 Load balancing example with three virtual connections

Figure 8.39 shows three screenshots from a simulation where three bi-directional virtual connections are established between node 0 and node 13. The colour of the

links depends on their usage. The mapping between link usage and link colour is shown in Table 8.5. The colour changes gradually when the link load changes, so if — for instance — the link load is 70%, the colour of the link will be orange (a mixture of red and yellow).

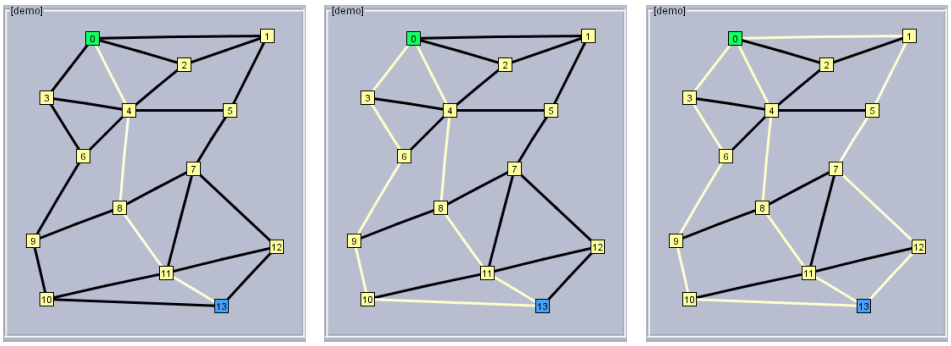
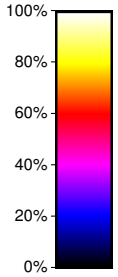


Figure 8.39 Load balancing example using three high-speed connections. Mode 1 is used for these simulations.

Table 8.5 Mapping between colour and link usage.

Colour	Usage
White	100%
Yellow	80%
Red	60%
Magenta	40%
Blue	20%
Black	0%



As shown in Figure 8.39a the first connection between node 0 and 13 goes through the nodes 4, 8, and 11 — i.e. the route with the lowest hop-count is chosen. The connection is set to use 95% of the maximum link speed, which leads to the (almost) white colour shown in Figure 8.39a.

Approximately 100 ms later, another connection is set up between node 0 and 13. Due to the high load of the route 0→4→8→11→13 the artificial ants have chosen another route for this connection, namely 0→3→6→9→10→13. The situation after the second connection has been set up is shown in Figure 8.39b — also this connection uses 95% of the maximum link speed.

Finally, after another 100 ms have passed, a third connection using 95% of the maximum link speed is established between node 0 and 13. This time, the artificial ants have chosen the shortest path capable of accepting the requested bandwidth. As shown in Figure 8.39c the third connection is assigned the route 0→1→5→7→12→13.

8.7.2 Load balancing example with multiple low-bandwidth virtual connections

The example in this section illustrates how traffic is distributed when multiple low-speed virtual connections are added between two nodes in a sample network. The simulations were carried out using mode 7 (dual-source ants) using the following line of actions:

1. During the first 100 ms no connections are added — this period is used to discover routes between all s-d pairs.
2. After 100 ms a the simulator start adding connections each using 10% of the capacity of a link in each direction between the nodes 0 and 1.
3. Every 33 ms a new connection (also using 10% of the capacity of a single link) is added between the nodes 0 and 1.
4. For each three added connections, the simulator removes two randomly chosen connections, in order to simulate a situation where calls are both added and terminated. This means that the total load between the nodes 0 and 1 increases by 10% of the capacity of a single link for each 100 ms.
5. The steps 3 and 4 are repeated over and over again, until the system can not place more connections between the nodes 0 and 1.

Figure 8.40 shows the network used for this load-balancing test as well as the load of the links after 1000, 1500, 2000, and 3600 milliseconds.

After 1000 ms (see Figure 8.40a) all connections are routed along the nodes 0→7→3→19→1. However, 500 ms later (Figure 8.40b) the ant controlled system prefers the route are routed along the nodes 0→4→2→16→1. Actually, at this time the new "favorite" route is more loaded than the old route, because many of the old connections have been removed. But still, the system is not at all using any of the non-shortest routes in the center of the network.

After 2000 ms the situation has changed considerably. Due to the high load of the two shortest paths, the route 0→6→10→14→18→1 has suddenly become the

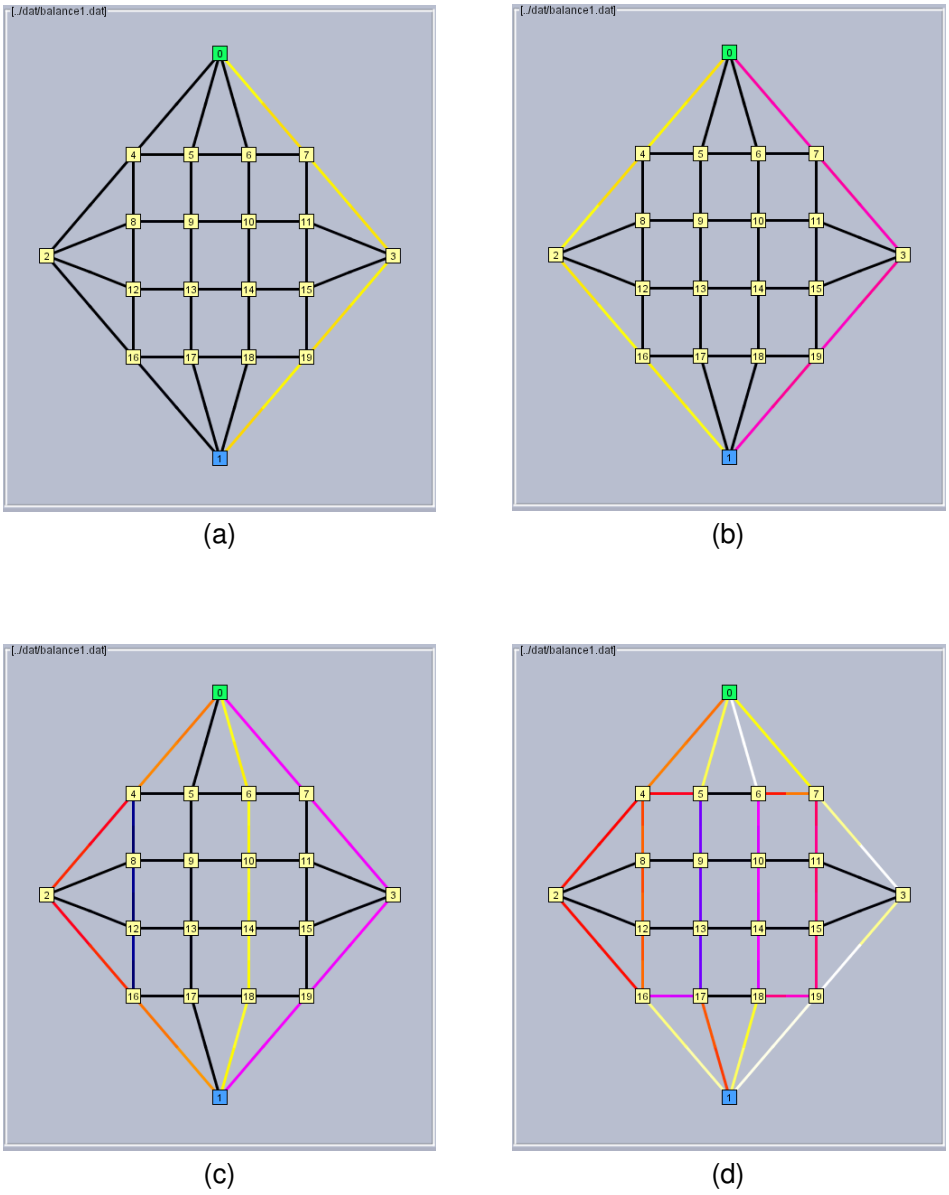


Figure 8.40 Load balancing example, where several low-speed connections are placed one by one between the nodes 0 and 1. The screenshots show the link load after (a) 1000 ms, (b) 1500 ms, (c) 2000 ms, and (d) 3600 ms.

best choice. Figure 8.40c shows the situation after 2000 ms where the new favorite route is actually more loaded than the two short routes.

Finally, at 3600 ms the situation is as shown in Figure 8.40d. The load is now almost equally distributed to all vertical links — only a few horizontal links carry any traffic. Actually, Figure 8.40d also shows the load of the links just before cells are beginning to get lost. If more connections are added, the buffers will overflow.

The total number of connections added to the network before the system became overloaded is:

$$\frac{3600-200}{100}+1=35$$

I.e. this corresponds to three fully loaded links and one link with a load of 50%. Obviously, since the nodes 0 and 1 have four ports, the theoretically maximum amount of traffic is 4 times the capacity of one link. However, the ants consume approximately 1–2% of the capacity, so there is actually only space for 9 connections in each of the four links, so the theoretically maximum number of connections between the nodes 1 and 2 using this test network is 36, which is very close to the 35 connections that could be added with the help of the artificial ants.

8.7.3 Load balancing example with traffic between two s-d pairs

In this section, connections are set up between the nodes 0 and 1 as described in Section 8.7.2, and in exactly the same way between the nodes 2 and 3. The same network as before is used for these tests, and again it is obvious that the theoretically maximum number of connections between the nodes 0 and 1 is 36 and the same limit applies to the maximum number of connections between the nodes 2 and 3.

Figure 8.41 shows the load of the links after 600, 1400, 2200, and 3500 milliseconds. In this simulation, things started getting problematic (calls were blocked and cells were lost in buffers) after 3500 ms. The number of connections set up between the nodes 0 and 1 at this critical moment was:

$$\frac{3500-200}{100}+1=34$$

Since the mechanism used for setting up connections between the nodes 2 and 3 followed the same rules, the number of connection between 2 and 3 is also 34 at this moment. This means that the total number of connections reached 2×34 (compared to the theoretical limit of 2×36 connections) before the network became overloaded.

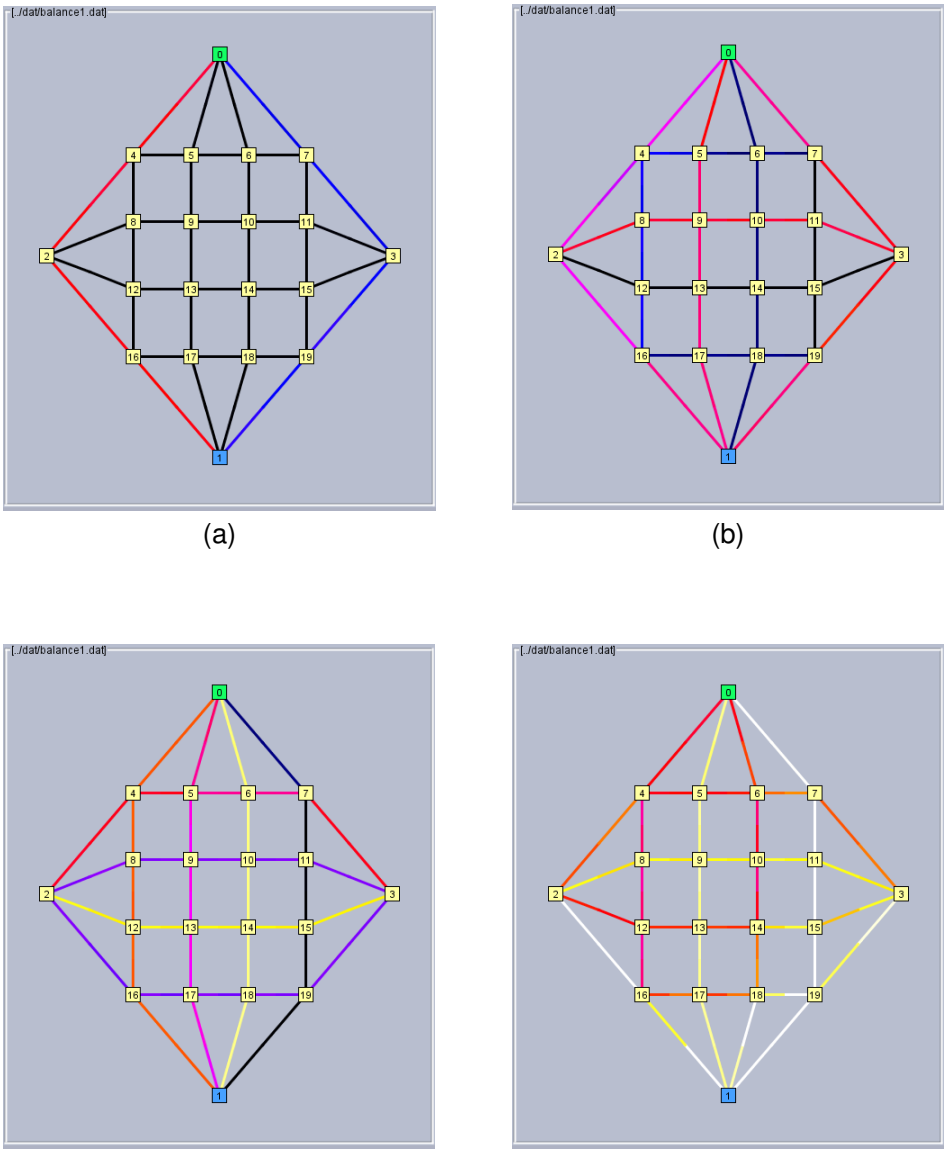


Figure 8.41 Load balancing example, where several low-speed connections are placed one by one between the nodes 0 and 1 and between the nodes 2 and 3, respectively. The screenshots show the link load after (a) 600 ms, (b) 1400 ms, (c) 2200 ms, and (d) 3500 ms.

8.7.4 Symmetrical versus asymmetrical traffic

The load balancing scenarios shown in the Sections 8.7.1, 8.7.2, and 8.7.3 all use symmetrical traffic. Actually, the types of articifial ants implemented in the simulator works best with symmetrically loaded links. The basic problem can be explained as follows:

1. Consider an extremely simple network comprising two nodes N and M connected by two links L1 and L2. L1 is highly loaded in the direction from N to M but only lightly loaded in the opposite direction. L2, on the other hand, does not carry any traffic at all.
2. Then, consider an artificial ant A travelling from node N to node M along the link L1.
3. Since the ant A travels along a highly loaded link, it will only leave a weak trail are mode M. Other ants arriving ant node M using the other link L2, will leave a stronger trail, so in the long run ants travelling from node M to N will prefer L2 over L1.
4. Now almost all ants arriving at node N have used link L2, so the trail left by these ants will, in turn, cause future ants travelling from N to M to prefer link L2 over L1.
5. The result: L1 will slowly become depleted even though it is only loaded in one direction.

Having said all this, it does not mean that the artificial ants are useless in situations with asymmetrical link load. The basic feedback mechanism, where ants travelling in one direction influence future ants travelling in the opposite direction, will always be a problem. However, in large backbone networks the traffic pattern will normally be more symmetrical than in small access networks, so is these types of networks the load balancing mechanisms of artificial ants will be particularly efficient.

8.8 Artificial ants for QoS routing

So far the feedback mechanisms of the artificial ants have been based on the age of the ants (including the penalty value when travelling along busy links) — i.e. the feedback from old ants is lower than the feedback from young ants. At any given moment there is only *one* route between each s-d pair which means that all types of traffic between any two nodes will be sent along the same route.

Supporting distinct treatment (from a routing point-of-view) of distinct traffic classes could be done by introducing different types of coexisting ants in the network. And in order to keep the deposited pheromone trails apart, each node should contain one pheromone table for each type of artificial ant.

Example: Consider a network support the following two traffic classes:

Best effort traffic For this traffic class the throughput is the most important parameter. The delay and delay variation are not particularly important here.

Real-time traffic The most important parameters for real-time are delay and delay variation.

An ant controlled network could support these two traffic classes by using two types of ants (and two pheromone tables in each node).

The first type of ants, suited for best effort traffic, should react mainly to the load of the links it travels along, and pay less attention to the delay caused by buffered or high-latency link (e.g. satellite links). This means that the *penalty* (see Section 8.1.4) should contribute more to the strength of the feedback than the actual age of the ant.

The other type of ants, suited for real-time traffic, should *not* pay too much attention to the link load. The overall delay should be the primary contributor to the strength of the feedback of this ant type, i.e. the *penalty* parameters should be considerably lower than for the "best effort ants".

When routing best effort calls, the selected route should of course be based on the "best effort" pheromone tables, and vice versa for real-time calls. If, for some reason, no route exists between an s-d pair according to the "real-time" pheromone tables, the node could choose to fall back to the "best effort" pheromone tables, even though the resulting route would not necessarily be capable of meeting the desired service request.

Naturally, proper routing is not enough to give adequate service guarantees. The buffers in the nodes should also support priority schemes so that real-time traffic can take precedence over best effort traffic.

A lot of other things — such as price, hop-count, delay variation, etc. — could be taken into account as well. However, further examination of QoS-enabled ants would be a topic for future work.

8.9 Scalability

One of the most important issues in almost any technical solution is scalability. Any solution that does not scale well with the size of the system will fail when

the system reaches a critical size. This section gives a short analysis about the scalability of systems using artificial ants (based on the model used in this chapter).

8.9.1 Memory consumption

Each node in an ant controlled network must hold a pheromone table to make the system work. The pheromone table has one column per port, that is m columns in total, and one row per node in the network (a total of n rows). This means that the memory consumption of each node scales like the following:

$$O(n \times m)$$

If the system uses different kinds of artificial ants in order to support several QoS classes, the memory consumption will instead scale like:

$$O(n \times m \times q)$$

q is the number of service classes.

However, for a node with a constant number of ports and a network with a constant number of service classes the resulting memory consumption becomes:

$$O(n \times m \times q) \in O(n)$$

8.9.2 Processing power

When a single-source artificial ant arrives at a node, the node must update one row in its pheromone table and possibly transmit the ant to another node. Since only one row in the pheromone table is touched for each arriving artificial ant, the total number of updates is m. Dual-source ants will require two updates in the pheromone table, so the total number of updates for this type of ants will be $2m$. Provided that the ant arrival rate is almost constant, the processing power when using single- and dual-source ants simply scales like:

$$O(m)$$

On the other hand, if a multi-source ant arrives at a node, multiple rows in the pheromone table will need to be updated. The number of rows requiring updates depends on the number of nodes visited by the ant before arriving at this node.

The average number of visited nodes will depend on the topology of the network. For instance, if the network consists of a string of n nodes, the average number of visited nodes for each artificial ant will be something like $\frac{n}{2}$. However, if all nodes can be reached directly from any other node, the average number of visited nodes will be close to 1. In short, the number of visited nodes depends greatly on the *network diameter*.

For a mesh network with n nodes organized as a $\sqrt{n}\times\sqrt{n}$ grid, the average number of visited nodes will not exceed $\sqrt{n}$. This means that when using multi-source ants for this type of network, the required processing power will scale like the following:

$$O(m \times \sqrt{n})$$

As already mentioned in Chapter 2 an algorithm like Dijkstra's shortest-path algorithm has a running time of $O(E \log V)$ per node for sparse networks[8]. Since the running time of Dijkstra's algorithm depends on other parameters than the processing power required by artificial ants, these two approaches are difficult to compare. However, it is quite clear that a system using single- or dual-source ants will scale better than a system using Dijkstra's algorithm. The required processing power for multi-source ants, on the other hand, depends greatly on the network topology.

8.10 Summary

This chapter has presented the results of an investigation of the use of artificial ants in telecommunication networks. Several topics have been touched, however the main focus has been on the following things:

Routing Routing with artificial ants has been done by selecting the trail with the strongest scent — i.e. routing with artificial ants is deterministic in contrast to the more or less random nature of ants.

Topology The influence of different topologies — such as ring, tree, and mesh — has been examined. The ants work with all of these topologies, however the best performance is obtained with mesh networks — and dense networks are better than sparse networks.

Fault management A lot of simulations have been made to investigate how well artificial ants can cope with network failures. Even in the most unlikely situations the artificial ants make an excellent job in finding alternative routes

[8] V is the number of nodes and E is the number of edges links in the network.

through the network. It typically takes in the order of 50–500 ms to find alternative routes in case of link failure.

Load balancing Artificial ants can be used for load balancing by simply implementing a mechanism discouraging ants to travel along too busy links. This is done by giving the ants a penalty, if they travel along busy links. Load balancing using artificial ants can easily cope with traffic patterns that change from second to second (or even faster). However, due to the trail laying mechanism of the implemented artificial ants (in the opposite direction of the motion of the ant) load balancing works best with symmetrical link loads.

QoS and artificial ants The current implementation only supports one service class. However, QoS-enabled ants could be implemented by using different kinds of ants that are susceptible to different network metrics such as load, delay, delay variation, cost, etc.

Scalability Artificial ants scales excellent (almost linearly) both in terms of memory consumption and in terms of required processing power.

Stability No stability issues have been noticed in *any* of the simulations made in this investigation, except for the case where *many* links vere either disabled or enabled simulataneously. And even in these rare cases, the system reached a stable state very quikly (in the order of 200-1000 ms).

Circuit-switched vs. packet-switched networks Even though the presented artificial ants can be used for circuit-switched as well as packet-switched networks, the deterministic route selection is indeed best suited for circuit-switched networks. Better support for packet switched protocols could be implemented by adding a random component to the route selection of datagrams and thereby make the routing of datagrams look like a cross between the deterministic routing used for circuit-switched protocols and the random walk of individual ants.

A major area in this chapter has been comparing the 10 different types of ants implemented in the simulator. A summary of the results is shown in Table 8.6 (the meaning of the modes are shown in Table 8.3). Some of the things shown in the table are based on the hands-on experience gained during this work — for instance, the ants' capabilities of discovering short or long routes are based entirely on observations.

Clearly, the size of single-source ants (modes 1 and 2) will be considerably smaller than ants remembering the route they have travelled (modes 3–6). Dual-source ants (mode 7–10), on the other hand, are almost as small as single-source ants. Table 8.6 also shows that in those situations where looping ants are killed (modes

Table 8.6 Comparison of the different ant modes. (○○○○○ is worst and ●●●●● is best.)

	Short route discovery	Long route discovery	Ant size	Convergence speed	Ant load	CPU scaling
Mode 1	●●●○○	●●○○○	●●●●●	●●○○○	●●○○○	●●●●●
Mode 2	●●○○○	●○○○○	●●●●●	●○○○○	●●●●●	●●●●●
Mode 3	●●○○○	●○○○○	●○○○○	●○○○○	●●●●○	●●●●○
Mode 4	●●●●●	●●●●●	●○○○○	●●●●●	●●●○○	●●●○○
Mode 5	●●●●○	●●●○○	●○○○○	●●●○○	●●●○○	●●○○○
Mode 6	●●●●●	●●●●○	●○○○○	●●●●○	●●●●○	●●●○○
Mode 7	●●●●●	●●●●○	●●●●○	●●●●○	●●●●○	●●●●●
Mode 8	●●●●○	●●●○○	●●●●○	●●●○○	●●●●●	●●●●○
Mode 9	●●●●●	●●○○○	●●●●○	●●●○○	●●●●○	●●●●●
Mode 10	●●●●○	●○○○○	●●●●○	●●○○○	●●●●●	●●●●○

1, 6, and 10) the ant load (i.e. the ant induced link load) is small, which of course is good. The required CPU power is best (smallest) for single-source and dual-source ants, whereas the required CPU power does not scale so well for multi-source ants. "Convergence speed" is the speed by with the system adjusts to new situations. I.e. this includes link error recovery, load balancing, the time it take to reach steady-state when new links are added, etc. The convergence speed is best for multi-source ants and the dual-source ants *not* using the pen-ultimate hop as their second virtual source (the modes 4–8).

The overall best performers based on Table 8.6 as well as the observations done during this work, are the modes 4 and 7. Actually, mode 4 has a slight edge over mode 7 in terms of speed, whereas mode 7 has the lead when it comes to ant size, ant load, and CPU scaling. All in all, it is difficult to announce a clear winner among the two modes, however mode 7 would be less complicated to implement compared to mode 4.

Even though artificial ants deal efficiently with a lot of problems in communication networks, it will be interesting to see whether they will ever be used in commercial networks. Since it is not possible to use exact mathematics on complex systems like networks controlled by artificial ants, it may never be possible to guarantee that artificial ant will not behave oddly under any (possibly obscure) circumstances, and this may turn out to be a major obstacle. However, networks also have to deal with dynamic and unpredictable traffic patterns (due to uncontrollable users), and this might indeed be a much bigger problem than the inherent probabilistic behaviour of ants. Also, the lack of understanding the machanisms in system like neural network has not stopped them from being used.

Chapter 9

Summary and Conclusion

This thesis has presented some of the basic mechanisms involved in resource management and routing in telecommunication networks. Resource management is about allocation, assignment, and provisioning of network resources in order to fulfil a requested service and to maintain the requested QoS. Routing — on the other hand — is normally not thought of as being part of resource management, however the utilization of the available network resources is highly dependent on the routing strategies used in the network.

In order to utilize the available network resources in an efficient way, exact and timely information about the network conditions must available. Obsolete information about the network resources may lead to non-optimal resource utilization or even to instabilities (e.g. oscillations).

After having described some of the basics of resource management, some of the most commonly used Internet technologies, such as IP, ATM, and MPLS, are described. IP is the most commonly used protocol for sending traffic between end-systems — i.e. it is used in access as well as core networks. ATM and MPLS — on the other hand — are mainly used in the backbone network as a solution for efficient transport of IP datagrams.

When ATM is used together with IP, the two layers are normally operated independently — i.e. the people controlling the IP network do not need to know much about the underlying ATM network and vice versa. The ATM layer does not really alleviate the IP layer from any of its heavy tasks — it simply does a good job forwarding IP packets from one IP router to another. The IP layer still has to deal with complex matters like efficient routing, providing limited service guarantees, etc.

With MPLS the story is a bit different. An MPLS-based network does indeed alleviate the IP-network from many of its complex tasks. The main idea behind MPLS is to do all complicated matters at the edge of the network (where the number of flows is manageable) thereby keeping the functionalities in the core part of the network as simple as possible. The number of flows in the core network is reduced considerably by making heavy use of traffic aggregation into so-called forward equivalence classes. Simply put, an MPLS network can be seen as one huge distributed IP router.

As already mentioned, routing has a major impact on the utilization of the resources in a network. The principles of the two predominant routing techniques used in today's networks — distance vector routing and link state routing — have been described and some of the routing protocols based on those two techniques are described too, such as OSPF, BGP, RIP, and PNNI (actually, PNNI is much more than simply a routing protocol).

All routing protocols have an upper limit to the size of the network they can handle. Therefore, the Internet — for instance — uses a hierarchical approach, where OSPF is typically used at the lowest level and BGP is used at the highest level. This hierarchical approach has some advantages and disadvantages. The most obvious advantage is, of course, that the size of each domain remains manageable. However, subdividing a network into many smaller domains also results in the loss of the "big picture". The larger the number of domains, the more difficult is gets to find globally optimal solutions — both in terms of routing and in terms of resource allocation.

The introduction to resource management and routing technologies is followed by the primary topics that have been investigated in this project. These are:

- Software agents
- Artificial ants

Software agents (or simply, *agents*) may — at their simplest — be seen as independent pieces of executing code capable of acting autonomously in an environment with expected and unexpected events. Software agents might also be mobile, however most agent implementations do not use mobile agents — generally speaking, mobility raises some serious security issues. The most important keywords for software agents are:

Autonomy The ability to operate with very limited human intervention or with no human intervention at all.

Social ability Mechanisms to communicate with other agents.

Reactivity Ability to respond quickly to changes in the environment.

Pro-activeness Goal-oriented behaviour.

In addition, it is often assumed that software agents are well-behaved and that they do not have conflicting goals.

This weak notion of agency has typically resulted in proprietary implementations of agent systems incapable of communicating with other agent systems. In order to overcome these interoperability issues, FIPA[1] was formed with the official goal of producing specifications making it easier to build agent systems that can communicate with foreign agent systems. Currently, FIPA has produced more than 40 documents covering topics such as agent management, agent communication language (ACL), ACL message structure, ACL message encoding and transport, interaction protocols, and much more.

The IMPACT project[2] described in this thesis was a European project with the intention of investigating the use of software agents in the telecommunication context. In essence, the IMPACT agents take over all the control and management functions and interface to the physical equipment (ATM terminals and switches) through two dedicated agents:

The proxy user agent interfaces directly with users through ATM UNI signalling (with a few extensions) on one side, and with the other IMPACT agents on the other side through a FIPA-like agent communication language.

The switch wrapper agent is capable of controlling the ATM switches in the network, either through SNMP or through the switch's API (if it is available).

The core agents in the IMPACT system implement a competitive marketplace model, where several service providers may bid for connections. It is assumed, that if a user has a contract with a specific service provider (SP), he also knows the tariff structure of that SP. Furthermore, if the user has contracts with several SPs, the proxy user agent (who acts on behalf of the user) assists the user in selecting the cheapest SP. In this selection process many things are taken into account like — for instance — the tariff structures (which might include volume discounts), the expected usage within one financial period, user preferences etc. I.e. an SP may be chosen even though it is more expensive than a competing SP, because it is expected to be cheaper in the long run.

[1] FIPA: Foundation for Intelligent Physical Agents

[2] IMPACT: **Imp**lementation of **a**gents for **C**AC on an ATM **t**estbed

Other than that, the IMPACT project has demonstrated scenarios where extra connections can be accepted by re-routing other connections or by transferring capacity between the so-called resource agents. Re-routing and capacity transfer are carried out during the call-establishment phase, and this typically results in a delay of a few seconds (in the scenarios used to test the system). Nevertheless, within the IMPACT project the most obvious advantage of using software agents, was the nice and clean interfaces they provided. Even though the agents were implemented in different locations (mainly London, Athens, and Copenhagen), interoperability was never an issue.

Of course the use of software agents in future telecommunication networks is not limited to the areas covered by the agents implemented in the IMPACT project. Especially, the use of *mobile* agents could have many new and interesting applications. For instance, a mobile agent acting on behalf of a user could move around from place to place and gather information about topics selected by the user, and then return with the results. Also e-commerce or Internet-based stock-exchange could be interesting applications of mobile agents.

Finally, it is important to stress that software agents do not by themselves provide any solutions. The most important thing when developing solutions for use in telecommunication networks is still a good knowledge of networks and typical network-related challenges so that common problems, such as stability and scalability issues, can be minimized. Also, for software agents to become a success in future telecommunication networks experience and an effort to produce standards are required. FIPA has already produced a lot of specifications making it easier to implement interoperable agents, however a similar effort has to be made in the context of telecommunication networks. Software agents will most likely "sneak" into all types of applications, and perhaps in some years developers will take the agent paradigm for granted, just like object-oriented techniques are a must today in many contexts.

The other topic that has been investigated in this project is the use of so-called *artificial ants* for things like routing, fault management, and load-balancing. Artificial ants borrow their behaviour from the trail laying mechanisms used by ants in nature. Ants in nature deposit pheromones (a kind of "footprints") as they walk around in their environment. These pheromones are sensed by other ants, and a trail with a strong "scent" has a higher probability of being chosen than a trail with a weak "scent". The result of this extremely simple behaviour is that ants can actually find the shortest paths (or near-shortest paths) between their nest and food sources scattered around in the environment. This exchange of information through the environment is called *stigmergy*.

In order to make the examinations of the use of artificial ants in telecommunication networks, a simulator was written based on a model by Schoonderwoerd et al. The basic model was extended to include a total of 10 different classes of artificial ants — some with a better behaviour than others. Also, in order to incite the artificial ants to quickly stop using busy links, a load-dependent penalty parameter was added to the model, and the result from this is that load-balancing works at much faster time-scales. Even though the implemented model *can* be used for connection-less networks, it works best with connection-oriented networks. The reason for this is that at any given moment only one route exists between two nodes (according to the ant model) and this means that the amount of traffic between two nodes can not exceed the capacity of a single link.

The simulations carried out on several ant controlled network scenarios[3] have shown that it typically takes in the order of 50–200 ms to go from the newly started system, where no routes are known between any source-destination pairs (sd-pairs), to a situation, where all sd-pairs can be interconnected through routes discovered by the artificial ants. However, the routes found at are not necessarily optimal. In fact, they will normally be far from optimal at first, but as time passes the routes get closer and closer to the optimum — this process will typically take a few seconds or less. Traditional routing schemes — on the other hand — try to optimize the routes right at the beginning. However, typical convergence times for routing schemes used in the Internet is in the order of minutes.

Also, the results have shown that excellent (or perhaps even, *unprecedented*) fault-management results can be obtained by using artificial ants. Even in cases where major parts of a network are destroyed, the artificial ants make an excellent job in adjusting to new and unexpected situations. For instance, single link failures are barely noticeable, since the artificial ants find alternative routes within a few milliseconds, and in case of multiple link failures all routes are typically restored within one second or faster.

The scalability of systems using artificial ants is very good. With the current model, the memory consumption scales linearly with the total number of nodes in the network, and the required processing power in each node scales linearly with the number of ports. This is certainly better than the scalability of link-state routing protocols, that are normally considered as being fast and efficient.

The results presented in this thesis suggest that artificial ants behave well with any imaginable network topology or condition. Naturally, this can not be seen as *conclusive* evidence, that artificial ants will not lead to instabilities under *any* circumstances. However, of all simulations carried out during this project, none of them exhibited any strange behaviour,

[3] Assuming that all nodes are ATM switches and that the links are running at 155 Mbit/s.

Finally, due to the excellent scalability, ants controlled networks can grow to very large sizes allowing a more flat network hierarchy than what is normally used in communication networks. A network controlled as one big entity allows for better resource utilization than the same network subdivided into multiple domains.

Based on the experience gained during this project with software agents and artificial ants, it is beyond any doubt that the use of these two approaches does indeed add a lot of benefits to telecommunication networks. Software agents are normally complex pieces of software, which make them unsuitable for tasks operating at extremely fast time-scales, whereas artificial ants are extremely simple, which in general makes them fast. This means that the obvious use of software agents in a network context would be in management-plane functions, whereas the obvious use of artificial ants would be in control-plane functions.

With these observations in mind it is obvious to suggest that artificial ants and software agents could be used *together* for controlling and managing telecommunication networks. In a routing and resource management context this means that software agents could take care of network engineering — i.e. putting the resources where traffic is, or where traffic is expected to be in the near future — whereas artificial ants could be in charge of routing, load balancing, and fault management.

Investigations of the synergy effects obtained when combining software agents and artificial ants will be hot topics for the future.

References

[1] ITU-T Recommendation I.113. *Vocabulary of terms for broadband aspects of IDSN*, June 1997.

[2] Kjør fort med bredbånd. `http://www.finnmark.net/it/artikler/191000-bredband.htm`.

[3] Zbigniew Dziong. *ATM Network Resource Management*. McGraw-Hill, 1997. ISBN 0-07-018546-8.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990. ISBN 0-262-03141-8.

[5] Lawrence G. Roberts. Explicit rate flow control — A 100 fold improvement over TCP, April 1997. `http://www.ziplink.net/~lroberts/Explicit-Rate/Explicit-Rate-Flow-Control.html`.

[6] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, IETF, January 2001.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, IETF, December 1998.

[8] ITU-T Recommendation G.707. *Network node interface for the synchronous digital hierarchy (SDH)*, October 2000.

[9] B. N. Bashforth and C. L. Williamson. Statistical multiplexing of self-similar video streams: Simulation study and performance results. In *Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 119–126. IEEE Computer Society, July 1998. ISBN 0-818-68566-2.

[10] Eva Gustafsson and Gunnar Karlsson. A literature survey on traffic dispersion. *IEEE Network*, 11(2):28–36, 1997.

[11] *Hobbes' Internet Timeline v5.6*, 2002. `http://www.zakon.org/robert/internet/timeline/`.

[12] Internet Protocol — DARPA Internet program protocol specification. RFC 791, IETF, September 1981.

[13] Latif Ladid. IPv6 — The new-generation Internet. *Ericsson Review No. 1*, pages 6–13, 2000.

[14] S. Deering. Host extensions for ip multicasting. RFC 1112, IETF, August 1989.

[15] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (CIDR) — an address assignment and aggregation strategy. RFC 1519, IETF, September 1993.

[16] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification. RFC 2460, IETF, December 1998.

[17] Rafael Osso, editor. *Handbook of Emerging Communications Technologies*. CRC Press LLC, 2000. ISBN 3-540-66350-9.

[18] R. Braden (editor), L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP), version 1 — functional specification. RFC 2205, IETF, September 1997.

[19] C. Hedrick. Routing information protocol. RFC 1058, IETF, June 1988.

[20] Frank Nielsen. *Graph Theory — Algorithms and Networks*. Department of Mathematics, Technical Univesity of Denmark, 1996.

[21] G. Malkin. RIP Version 2. RFC 2453, IETF, November 1998.

[22] John Kane, editor. *Internetworking Technologies Handbook*. Cisco Press, 3rd edition, 2000. ISBN 1-58705-001-3.

[23] J. Moy. OSPF Version 2. RFC 2328, IETF, April 1998.

[24] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, IETF, March 1995.

[25] A. Fosgerau. *Terminals and Networks for Multimedia Applications*. PhD thesis, Research Center COM, Technical University of Denmark, July 1999.

[26] C. Labovitz, R. Jahanian, and Malan F. Internet routing instability. *IEEE ACM Transactions on Networking*, 6(5):515–528, October 1998.

[27] ITU-T Recommendation I.361. *B-IDSN ATM Layer Specification*, February 1999.

[28] ITU-T Recommendation Q.2931. *B-IDSN Application Protocols for Access Signalling*, February 1995.

[29] The ATM Forum. *ATM User-Network Interface (UNI) Signalling Specification Version 4.0*, July 1996. `af-sig-0061.000`.

[30] Jörgen Axell and Fiffi Hellstrand. ATM traffic management and resource optimization. *Ericsson Review No. 1*, pages 18–23, 1998.

[31] The ATM Forum. *Interim Inter-switch Signaling Protocol (IISP) Specification v1.0*, December 1994. `af-pnni-0026.000`.

[32] The ATM Forum. *Private Network-Network Interface Specification Version 1.0 (PNNI 1.0)*, March 1996. `af-pnni-0055.000`.

[33] Göran Hågård and Mikael Wolf. Multiprotocol label switching in ATM networks. *Ericsson Review No. 1*, pages 32–39, 1998.

[34] Grenville Armitage. MPLS: The magic behind the myths. *IEEE Communications Magazine*, 38(1):124–131, January 2000.

[35] T.M. Chen and T.H. Oh. Reliable services in MPLS. *IEEE Communications Magazine*, 37(12):58–62, December 1999.

[36] Vu Anh Pham and Ahmed Karmouch. Mobile software agents: An overview. *IEEE Communications Magazine*, 36(7):26–37, 1998.

[37] R.H. Glitho, E. Olougouna, and S. Pierre. Mobile agents and their use for information retrieval: A brief overview and an elaborate case study. *IEEE Network*, 16(1):34–41, 2002.

[38] M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Eng. Rev.*, Vol. 10, Issue 2:115–152, 1995.

[39] D. van Thanh. Using mobile agents in telecommunications. In A.M. Tjoa and R.R. Wagner, editors, *12th International Workshop on Database and Expert Systems Applications, 2001*, pages 685–688, September 2001.

[40] S. Steward and S. Appleby. Mobile software agents for control of distributed systems based on principles of social insect behaviour. In C. S. Ng and T. S. Yeo, editors, *Singapore ICCS '94. Conference Proceedings.*, volume 2, pages 549–53. IEEE, 1994. ISBN-0-7803-2046-8.

[41] Eric Bonabeau, Florian Henaux, Sylvain Guerin, Dominique Snyers, Pascale Kuntz, and Guy Theraulaz. Routing in telecommunications networks with ant-like agents. *Lecture Notes in Computer Science*, 1437:60–71, 1998.

[42] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ants for load balancing in telecommunications networks. Technical report, HP Labs, March 1996.

[43] Michael J. Wooldridge and Nicholas R. Jennings. Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computing*, Vol. 3, Issue 3:20–27, 1999.

[44] S.J. Poslad, R.A. Bourne, Hayzelden A.L.G., and P. Buckle. *Agent Technology for Communication Infrastructure*, chapter 1: An Introduction. John Wiley & Sons, Ltd., 2001. ISBN 0-471-49815-7.

[45] Joseph P. Bigus and Jennifer Bigus. *Constructing Intelligent Agents With Java*. John Wiley & Sons, December 1997.

[46] O. Krone, B.T. Messmer, H. Almiladi, and T. Curran. *Agent Technology for Communication Infrastructure*, chapter 3: Java Framework for Negotiating Management Agents. John Wiley & Sons, Ltd., 2001. ISBN 0-471-49815-7.

[47] The web robots pages. `http://www.robotstxt.org/wc/robots.html`.

[48] T. Finin, R. Fritzon, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM)*. ACM Press, 1994.

[49] Foundation for Intelligent Physical Agents. *FIPA Agent Management Specification*, October 2001.
`http://www.fipa.org/specs/fipa00023/`.

[50] IBM. *IBM Aglets Software Development Kit*. `http://www.trl.ibm.com/aglets/`.

[51] Nortel Networks. *FIPA-OS*, 2002. `http://www.nortelnetworks.com/fipa-os`.

[52] University of Cincinnati. *JAFMAS — Java-Based Framework for Multi-Agent Systems*.
`http://www.ececs.uc.edu/ abaker/JAFMAS/`.

[53] Stanford University. *JATLite — Java Agent Template, Lite*.
`http://java.stanford.edu/`.

[54] SRI. *Open Agent Architecture*. `http://java.stanford.edu/`.

[55] Foundation for Intelligent Physical Agents. *FIPA*. `http://www.fipa.org/`.

[56] Foundation for Intelligent Physical Agents. *FIPA — Statement of Intent*.
`http://www.fipa.org/about/mission.html`.

[57] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Service Specification*, August 2001. `http://www.fipa.org/specs/fipa00067/`.

[58] Foundation for Intelligent Physical Agents. *FIPA ACL Message Representation in String Specification*, August 2001. `http://www.fipa.org/specs/fipa00070/`.

[59] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, December 2001. Revision 2.6.

[60] Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, August 2001. `http://www.fipa.org/specs/fipa00061/`.

[61] Foundation for Intelligent Physical Agents. *FIPA Communicative Act Library Specification*, August 2001. `http://www.fipa.org/specs/fipa00037/`.

[62] Foundation for Intelligent Physical Agents. *FIPA SL Content Language Specification*, August 2001. `http://www.fipa.org/specs/fipa00008/`.

[63] Foundation for Intelligent Physical Agents. *FIPA CCL Content Language Specification*, August 2001. http://www.fipa.org/specs/fipa00009/.

[64] Foundation for Intelligent Physical Agents. *FIPA KIF Content Language Specification*, August 2001. http://www.fipa.org/specs/fipa00010/.

[65] Foundation for Intelligent Physical Agents. *FIPA RDF Content Language Specification*, August 2001. http://www.fipa.org/specs/fipa00011/.

[66] Foundation for Intelligent Physical Agents. *FIPA ACL Message Representation in Bit-Efficient Encoding Specification*, August 2001. http://www.fipa.org/specs/fipa00069/.

[67] Foundation for Intelligent Physical Agents. *FIPA ACL Message Representation in XML Specification*, August 2001. http://www.fipa.org/specs/fipa00071/.

[68] Foundation for Intelligent Physical Agents. *FIPA Interaction Protocol Library Specification*, August 2001. http://www.fipa.org/specs/fipa00025/.

[69] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Protocol for IIOP Specification*, August 2001. http://www.fipa.org/specs/fipa00075/.

[70] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Protocol for WAP Specification*, August 2001. http://www.fipa.org/specs/fipa00076/.

[71] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Protocol for HTTP Specification*, August 2001. http://www.fipa.org/specs/fipa00084/.

[72] A.C. Price, L.G. Cuthbert, and M. Potts. Definition of collaboration. Deliverable 000, ACTS-IMPACT, A0324, June 1998.

[73] J. Bigham, H.-P. Gisiger, B. Messmer, J. Soldatos, and E. Vayias. Specification of scenarios. Deliverable 001, ACTS-IMPACT, A0324, June 1998.

[74] J. Bigham, L.G. Cuthbert, Z. Luo, E. Vayias, H. Almiladi, and B. Messmer. Specification of agents. Deliverable 003, ACTS-IMPACT, A0324, August 1998.

[75] E. Vayias, M.S. Hansen, J. Soldatos, and P.V. Jensen. Specification of signalling implementation. Deliverable 004, ACTS-IMPACT, A0324, October 1998.

[76] J. Soldatos and A.C. Price. Signalling available. Deliverable 006, ACTS-IMPACT, A0324, September 1999.

[77] E. Vayias, J. Soldatos, and L.G. Cuthbert. Definition of performance criteria. Deliverable 008, ACTS-IMPACT, A0324, January 2000.

[78] J. Bigham, Z. Luo, L.G. Cuthbert, M.S. Hansen, and P.V. Jensen. Report on agent capabilities. Deliverable 009, ACTS-IMPACT, A0324, March 2000.

[79] A.C. Price and L.G. Cuthbert. Integrated system demonstration. Deliverable 010, ACTS-IMPACT, A0324, February 2000.

[80] Sun Microsystems. *Java™ 2 Platform Standard Edition v. 1.4 — Performance and Scalability Guide*. http://java.sun.com/j2se/1.4/performance.guide.html.

[81] Gopalan Suresh Raj. *A Detailed Comparison of CORBA, DCOM and Java RMI*, 2000. http://gsraj.tripod.com/misc/compare.html.

[82] The ATM Forum. *ATM User-Network Interface Version 3.1 (UNI 3.1) Specification*, July 1994.

[83] M. Hansen, P. Jensen, J. Soldatos, and E. Vayias. *Agent Technology for Communication Infrastructure*, chapter 12: Low-Level Control of Network Elements from an Agent Platform. John Wiley & Sons, Ltd., 2001. ISBN 0-471-49815-7.

[84] M. Hansen, P. Jensen, J. Soldatos, and E. Vayias. Low-level control of network elements from an agent platform. In Alex L. G. Hayzelden, editor, *IMPACT'99: Communication Networks and the IMPACT of Software Agents*, pages 23–30, December 1999. ISBN 0-904-18864-7.

[85] Mads S. Hansen, John K. Soldatos, and Evangelos K. Vayias. Low-level control of ATM network elements using agent technology. In *SCI'2000/ISAS'2000 — Information Systems, Analysis, and Synthesis/Systemics, Cybernetics, and Informatics*. The International Institute of Informatics and Systemics, July 2000.

[86] M. Laubach and J. Halpern. Classical IP and ARP over ATM. RFC 2225, IETF, April 1998.

[87] The ATM Forum. *LAN Emulation Over ATM Version 1.0*, January 1995. `af-lane-0021.000`.

[88] The ATM Forum. *Multi-Protocol Over ATM Version 1.1*, May 1999. `af-mpoa-0114.000`.

[89] ITU-T Recommendation Q.2100. *B-IDSN Signalling ATM Adaptation Layer (SAAL) Overview Description*, July 1994.

[90] *ATM on Linux*. `http://linux-atm.sourceforge.net/`.

[91] M.S. Hansen and P.V. Jensen. Software available for integration. Milestone M13, ACTS-IMPACT, A0324, April 1999.

[92] A. Costoloni, A.C. Price, L.G. Cuthbert, and M. Potts. Specification of the enhanced platform. Deliverable 002, ACTS-IMPACT, A0324, June 1998.

[93] Agorics, Inc. *Auctions: Going, Going, Gone! A Survey of Auction Types*, 1996. `http://www.agorics.com/Library/auctions.html`.

[94] Ole Groth Jørsboe. *Sandsynlighedsregning*. Matematisk Institut, Technical University of Denmark, 1992.

[95] Xiaojun Fang, Rainer Iraschko, and Rohit Sharma. All-optical four-fiber bidirectional line-switched ring. *Journal of Lightwave Technology*, 17(8):1302–1308, 1999.

[96] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary. The impact of Internet policy and topology on delayed routing convergence. *INFOCOM 2001. Proceedings. IEEE*, pages 537–546, April 2001. ISBN 0-780-37016-3.

Appendix A

List of Acronyms

A

AAL ATM Adaption Layer

ABR Available Bit-Rate

ABT ATM Block Transfer

ACC Agent Communication Channel

ACL Agent Communication Language

ACTS Advanced Communications Technology and Services

ADSL Asymmetric Digital Subscriber Line

AI Artificial Intelligence

AID Agent Identifier

AMS Agent Management System

AP Agent Platform

API Application Programming Interface

ATM Asynchronous Transfer Mode

B

BGP Border Gateway Protocol

BHLI Broadband High-Layer Information

B-ISDN Broadband ISDN

BLLI Broadband Low-Layer Information

BLSR Bi-directional Line-Switched Ring

BT Burst Tolerance

C

CA Connection Agent

CAC Connection Admission Control

CBR Constant Bit Rate

CDR Common Data Representation

CDV Cell Delay Variation

CDVT Cell Delay Variation Tolerance

CLIP Classical IP (over ATM)

CLP Cell Loss Priority

CLR Cell Loss Rate

COM Component Object Model

CORBA Common Object Request Broker Architecture

CPE Customer Premises Equipment

CPN Customer Premises Network

CRC Cyclic Redundancy Check

CTD Cell Transfer Delay

D

DBR Deterministic Bit-Rate

DCOM Distributed COM

DF Directory Facilitator

DoS Denial of Service

DSL Digital Subscriber Line

DV Distance Vector

DVD Digital Versatile Disc

E

EGRP Exteriour Gateway Routing Protocol

F

FDDI Fiber Distributed Data Interface

FEC Forwarding Equivalence Class

FIFO First In, First Out

FIPA Foundation for Intelligent Physical Agents

G

GFC Generic Flow Control

GIOP General Inter-ORB Protocol

GUI Graphical User Interface

H

HAP Home Agent Platform

HEC Header Error Control

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

I

IDL Interface Definition Language

IGRP Interiour Gateway Routing Protocol

IIOP Internet Inter-ORB Protocol

IMPACT Implementation of Agents for CAC on an ATM Testbed

IP Internet Protocol / Interaction Protocol (agents)

ISDN Integrated Services Digital Network

ISO International Organization for Standardization

ISP Internet Service Provider

IST Information Society Technologies

ITU International Telecommunication Union

ITU-T Telecommunication of ITU

J

J2EE Java 2 Enterprise Edition

JAM Java Agent Template

JDBC Java Database Connection

JIT Just In Time

JNI Java Native Interface

JRE Java Runtime Environment

K

KIF Knowledge Interchance Format

KQML Knowledge Query and Manipulation Language

L

LAN Local Area Network

LANE Local Area Network Emulation

LDP Label Distribution Protocol

LS Link State

LSP Label Switched Path

LSR Label Switched Router

M

MPEG Motion Pictures Expert Group

MPLS Multiprotocol Label Switching

MPOA Multiprotocol Over ATM)

MTP Message Transport Protocol

MTS Message Transport Service

N

NE Network Element

NIC Network Interface Card

NNI Network Network Interface

NP Network Provider

NPA Network Provider Agent

NSP Network Service Provider

O

OAM Operation Administration and Maintenance

OMG Object Management Group

OO Object Oriented

ORB Object Request Broker

OS Operating System

OSI Open Systems Interconnection

OSPF Open Shortest-Path First

P

PC Personal Computer

PCA Proxy Connection Agent

PCR Peak Cell Rate

PDA Personal Digital Assistant

PDF Portable Document Format

PDH Plesiochronous Digital Hierarchy

PHY Physical Layer

PLAF Pluggable Look-And-Feel

PON Passive Optical Network

POTS Plain Old Telephone Service

PSTN Public Switched Telephone Network

PTI Payload Type Identifier

PUA Proxy User Agent

PVC Permanent Virtual Connection

Q

QoS Quality of Service

R

RA Resource Agent

RIP Routing Information Protocol

RMI Remote Method Invocation

S

SAAL Signalling AAL

SAP Service Access Point

SBR Statistical Bit-Rate

SCR Sustainable Cell-Rate

SDH Synchronous Digital Hierarchy

SDK Software Development Kit

SDL Specification and Description Language

SNMP Simple Network Management Protocol

SONET Synchronous Optical Network

SP Service Provider

SPA Service Provider Agent

SSP Secondary Service Provider

SwWA Switch Wrapper Agent

T

TCP Transfer Control Protocol

ToS Type of Service

U

UML Unified Modeling Language (OMG standard)

UMTS Universal Mobile Telecommunications System

UNI User Network Interface

UPC Usage Parameter Control

URL Uniform Resource Locator

V

VBR Variable Bit Rate

VC Virtual Channel

VCC Virtual Channel Connection

VCI Virtual Channel Identifier

VP Virtual Path

VPC Virtual Path Connection

VPI Virtual Path Identifier

W

W3C World Wild Web Consortium

WAP Wireless Application Protocol

WDM Wavelength Division Multiplexing

WWW World Wide Web

X

XHTML Reformulation of HTML 4 in XML

XML Extensible Markup Language

Appendix B

FIPA performatives

Table B.1 shows all 22 performatives (also called "communicative acts") defined by FIPA. For a more formal definition see the *FIPA Communicative Act Library Specification* [61].

Table B.1 FIPA performatives and short description.

FIPA performative	Functional summary
`Accept Proposal`	The action of accepting a previously submitted proposal to perform an action.
`Agree`	The action of agreeing to perform some action, possibly in the future.
`Cancel`	The action of one agent informing another agent that the first agent no longer has the intention that the second agent perform some action.
`Call for Proposal (cfp)`	The action of calling for proposals to perform a given action.
`Confirm`	The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.
`Disconfirm`	The sender informs the receiver that a given proposition is false, where the receiver is known to believe, or believe it likely that, the proposition is true.

(Continued...)

(Continued from previous page)

FIPA performative	Functional summary
`Failure`	The action of telling another agent that an action was attempted but the attempt failed.
`Inform`	The sender informs the receiver that a given proposition is true.
`Inform If`	A macro action for the agent of the action to inform the recipient whether or not a proposition is true.
`Inform Ref`	A macro action for the sender to inform the receiver the object which corresponds to a descriptor, for example, a name.
`Not Understood`	The sender of the act (for example, *i*) informs the receiver (for example, *j*) that it perceived that *j* performed some action, but that *i* did not understand what *j* just did. A particular common case is that *i* tells *j* that *i* did not understand the message that *j* has just sent to *i*.
`Propagate`	The sender intends that the receiver treat the embedded message as sent directly to the receiver, and wants the receiver to identify the agents denoted by the given descriptor and send the received *propagate* message to them.
`Propose`	The action of submitting a proposal to perform a certain action, given certain conditions.
`Proxy`	The sender wants the receiver to select target agents denoted by a given description and to send an embedded message to them.
`Query If`	The action of asking another agent whether or not a given proposition is true.
`Query Ref`	The action of asking another agent for the object referred to by a referential expression.
`Refuse`	The action of refusing to perform a given action, and explaining the reason for the refusal.

(Continued...)

(Continued from previous page)

FIPA performative	Functional summary
`Reject Proposal`	The action of rejecting a proposal to perform some action during negotiation.
`Request`	The sender requests the receiver to perform some action. One important class of uses of the request act is to request the receiver to perform another communicative act.
`Request When`	The sender wants the receiver to perform some action when some given proportion becomes true.
`Request Whenever`	The sender wants the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.
`Subscribe`	The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

Appendix C

IMPACT performance measurements

C.1 The Poisson distribution

The Poisson distribution function for three different values of λ is shown in Figure C.1 (solely for comparison reasons.)

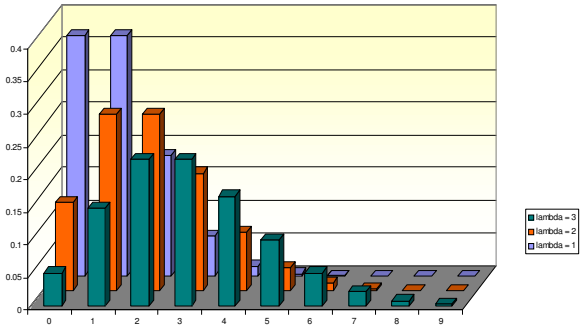


Figure C.1 The Poisson distribution for three values of λ.

C.2 Dummy switch wrapper agents

The Figures C.2, C.3, C.4 C.5, and C.6 show the time (in milliseconds) it takes to set up ATM connections using dummy switch wrapper agents (SwWAs). These measurements are very suitable for finding out how much time the Java agents consume *without* being slowed down by configuring the ATM switches through the management interface (through SNMP).

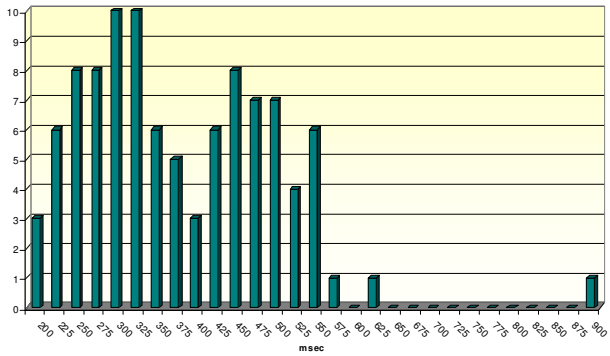


Figure C.2 Time to complete **loopback** connection setup using **dummy** switch wrapper agents (SwWAs) — 100 runs.

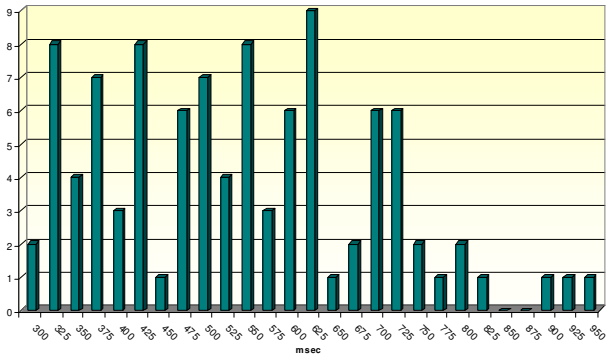


Figure C.3 Time to complete **local** connection setup using **dummy** switch wrapper agents (SwWAs) — 100 runs.

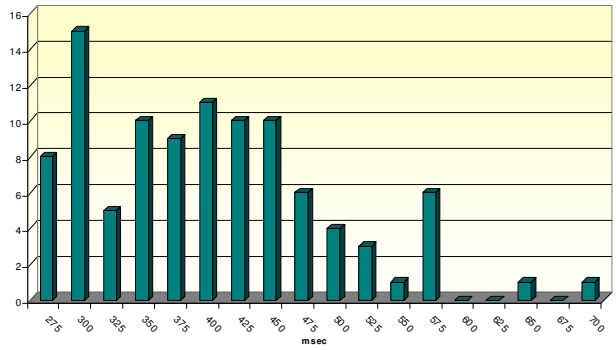


Figure C.4 Time to complete **point-to-point** connection setup using **dummy** switch wrapper agents (SwWAs) — 100 runs.

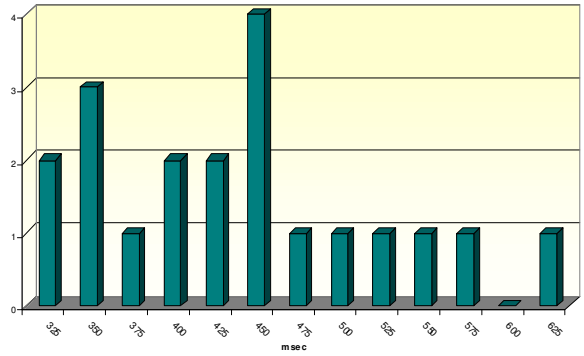


Figure C.5 Time to perform **capacity transfer** and to complete a connection setup using **dummy** switch wrapper agents (SwWAs) — 20 runs.

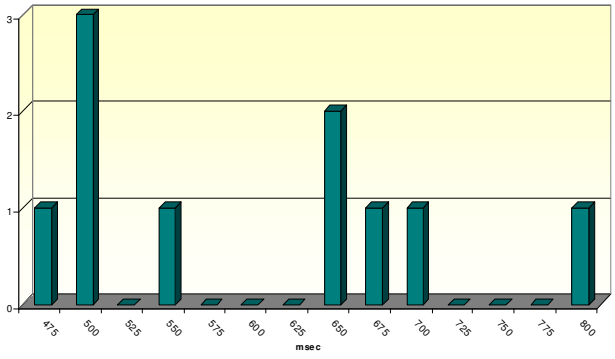


Figure C.6 Time to **re-route** another connection and to complete a connection setup using **dummy** switch wrapper agents (SwWAs) — 10 runs.

C.3 Real switch wrapper agents

The measurements in this section — the Figures C.7, C.8, C.9, C.10, and C.11 — are made using the "real" SwWAs. This means the results *include* the time it takes to configure the ATM switches through SNMP.

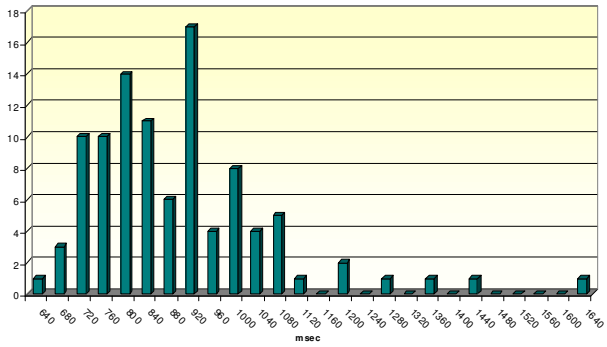


Figure C.7 Time to complete **loopback** connection setup using **real** switch wrapper agents (SwWAs) — 100 runs.

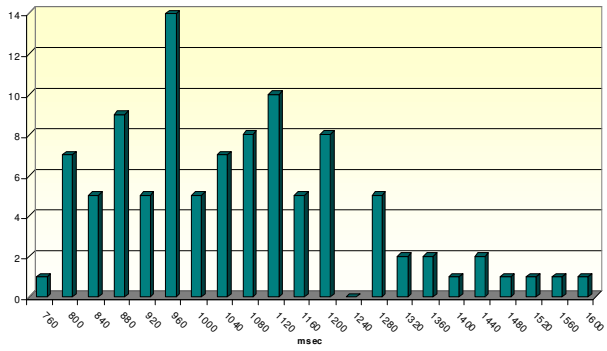


Figure C.8 Time to complete **local** connection setup using **real** switch wrapper agents (SwWAs) — 100 runs.

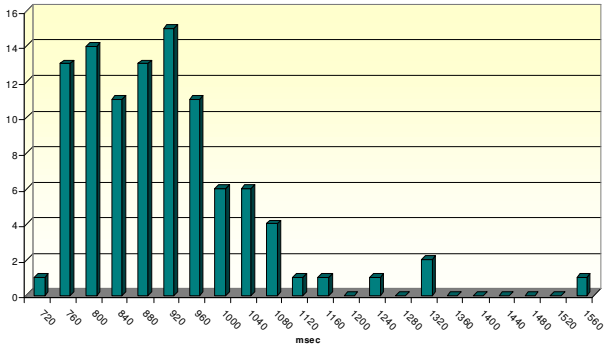


Figure C.9 Time to complete **point-to-point** connection setup using **real** switch wrapper agents (SwWAs) — 100 runs.

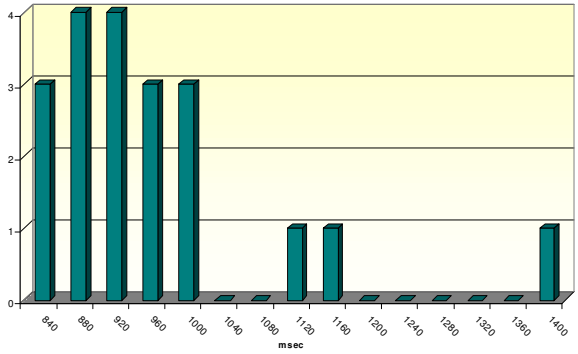


Figure C.10 Time to perform **capacity transfer** and to complete a connection setup using **real** switch wrapper agents (SwWAs) — 20 runs.

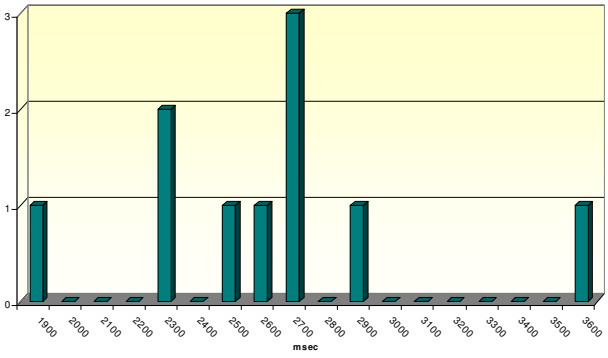


Figure C.11 Time to **re-route** another connection and to complete a connection setup using **real** switch wrapper agents (SwWAs) — 10 runs.

C.4 Fore ASX-200 switch

The Figures C.12, and C.13 show the time it takes to set up connections using ATM UNI signalling (version 3.1) — i.e. without any software agents. The are no measurements for point-to-point connections due to the lack of a PNNI-capable testbed.

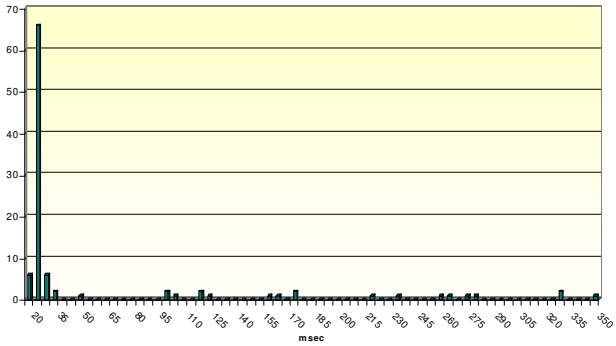


Figure C.12 Time to complete **loopback** connection setup using **UNI signalling** with a Fore ASX-200 switch.

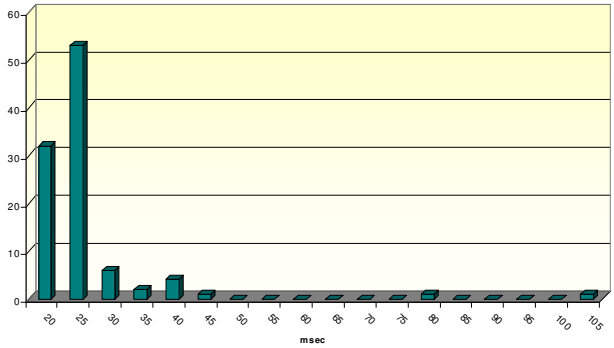


Figure C.13 Time to complete **local** connection setup using **UNI signalling** with a Fore ASX-200 switch.

Appendix D

Artificial Ants — Simulation Results

This appendix contains a lot of results from simulating how artificial ant behave in networks. The results presented here are only commented briefly and not discussed. See Chapter 8 for a discussion of the results.

D.1 Simulation modes

As explained in Chapter 8 the simulator is capable of using different types of ants. For convenience, the mapping between simulation mode number and the simulation settings is repeated in Table D.1. The modes 11 and 12 are not used at all, since they are in function identical to the modes 1 and 2, respectively.

D.2 Artificial ants and network topology

For all the simulations in this section the initial (time, $t = 0$) values in each of the pheromone tables were all equal (i.e. a totally clean network). Furthermore, all the results presented in this section are averaged of 100 runs in order to compensate for the random behaviour of each ant and to get smoother curves.

D.2.1 Mesh topology

Figure D.1 on page 187 shows the number of discovered routes between all possible combinations of s-d pairs as well as average hop-count of the routes discovered

Table D.1 Mapping between mode numbers and simulation settings.

Mode	Simulation settings
1.	Simple ants (no options selected)
2.	Simple ants + Kill looping ants
3.	Smart ants + Send back looping ants
4.	Smart ants + Multi-source ants
5.	Smart ants + Multi-source ants + Send back looping ants
6.	Smart ants + Multi-source ants + Kill looping ants
7.	Smart ants + Dual-source ants
8.	Smart ants + Dual-source ants + Kill looping ants
9.	Smart ants + Dual-source ants + Use penultimate hop as source
10.	Smart ants + Dual-source ants + Use penultimate hop as source + Kill looping ants
11.	Smart ants
12.	Smart ants + Kill looping ants

so far using a mesh network. The simulations have been made for all the modes 1–10.

The link load induced by artificial ants is illustrated in Figure D.2 on page 188. The average load as well as the maximum load is shown. Please note that the Y-axis uses different scales for the average and maximum load, respectively.

D.2.2 Tree topology

Figure D.3 on page 189 contains the number of discovered routes and the average hop-count as a function of time. The average load and the maximum load induced by the artificial ants are shown in Figure D.4 on page 190.

D.2.3 Ring topology

Figure D.5 on page 191 shows the number of discovered routes and the average hop-count of the routes as a function of time using a ring network. Furthermore, the ant induced load (average and maximum) is shown if Figure D.6 on page 192.

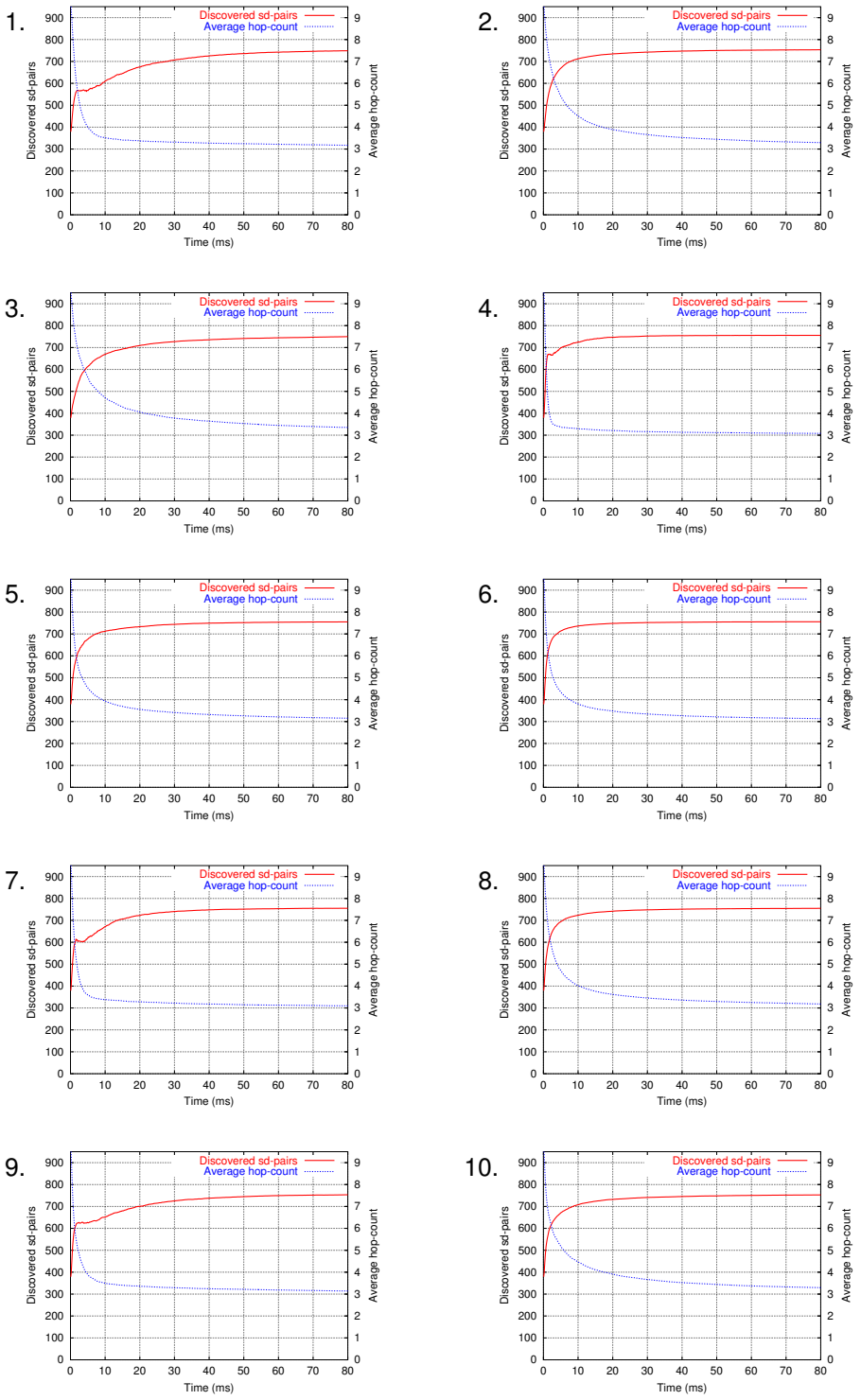


Figure D.1 The number of discovered routes between s-d pair and the average hop-count of the discovered routes using the **mesh** topology.

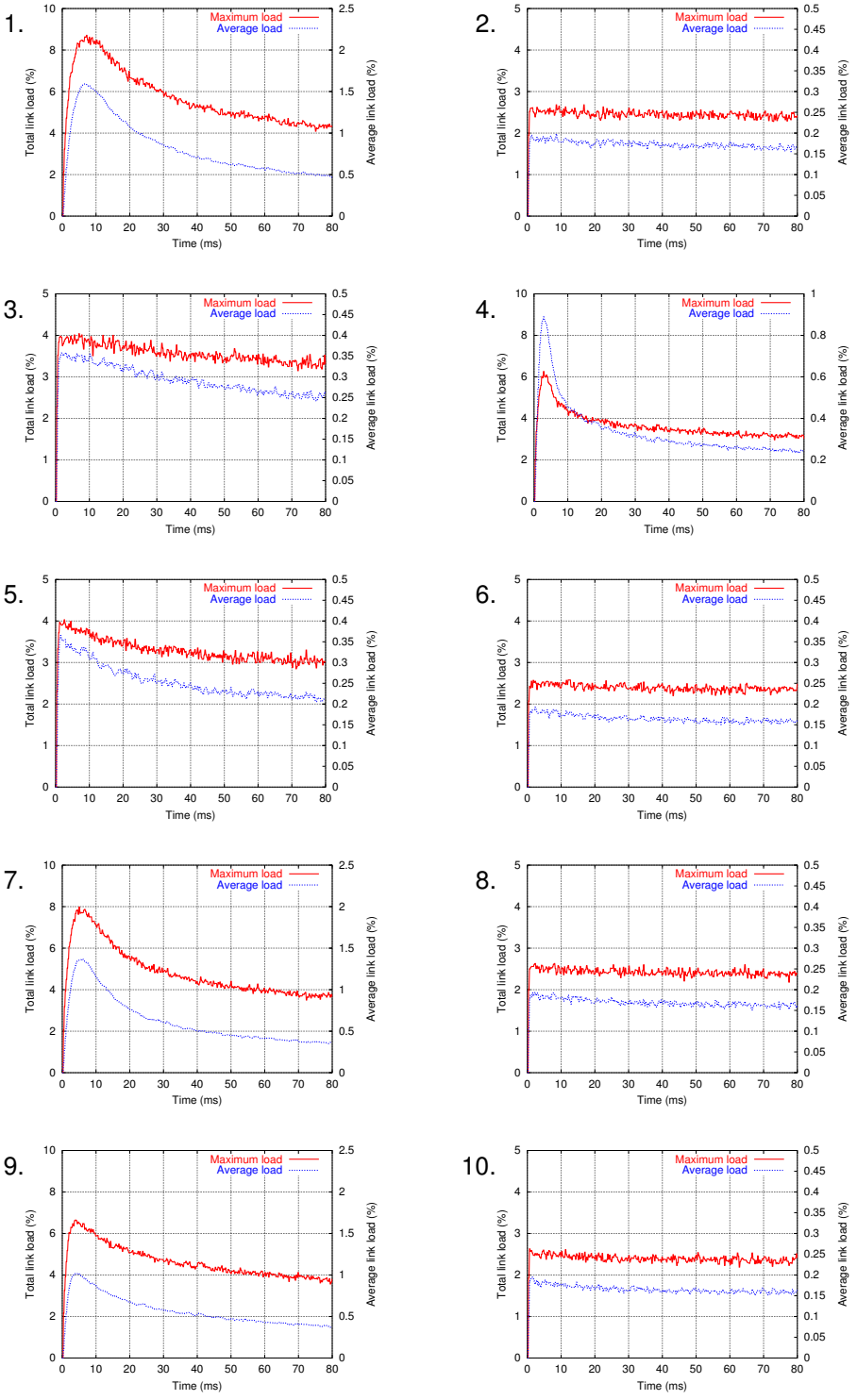


Figure D.2 Maximum and average load induced by the artificial ants using the **mesh** topology.

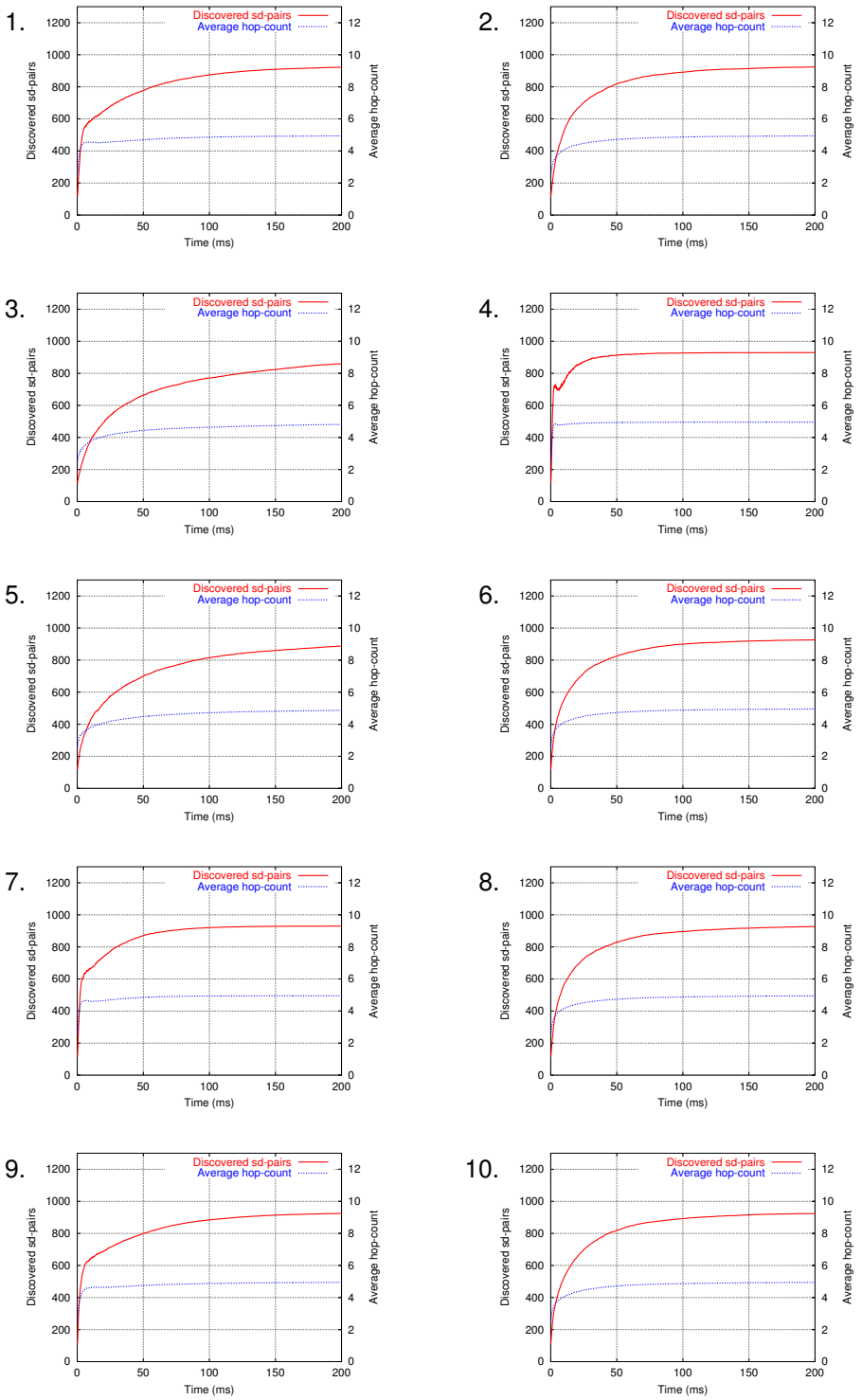


Figure D.3 The number of discovered routes between s-d pair and the average hop-count of the discovered routes using the **tree** topology.

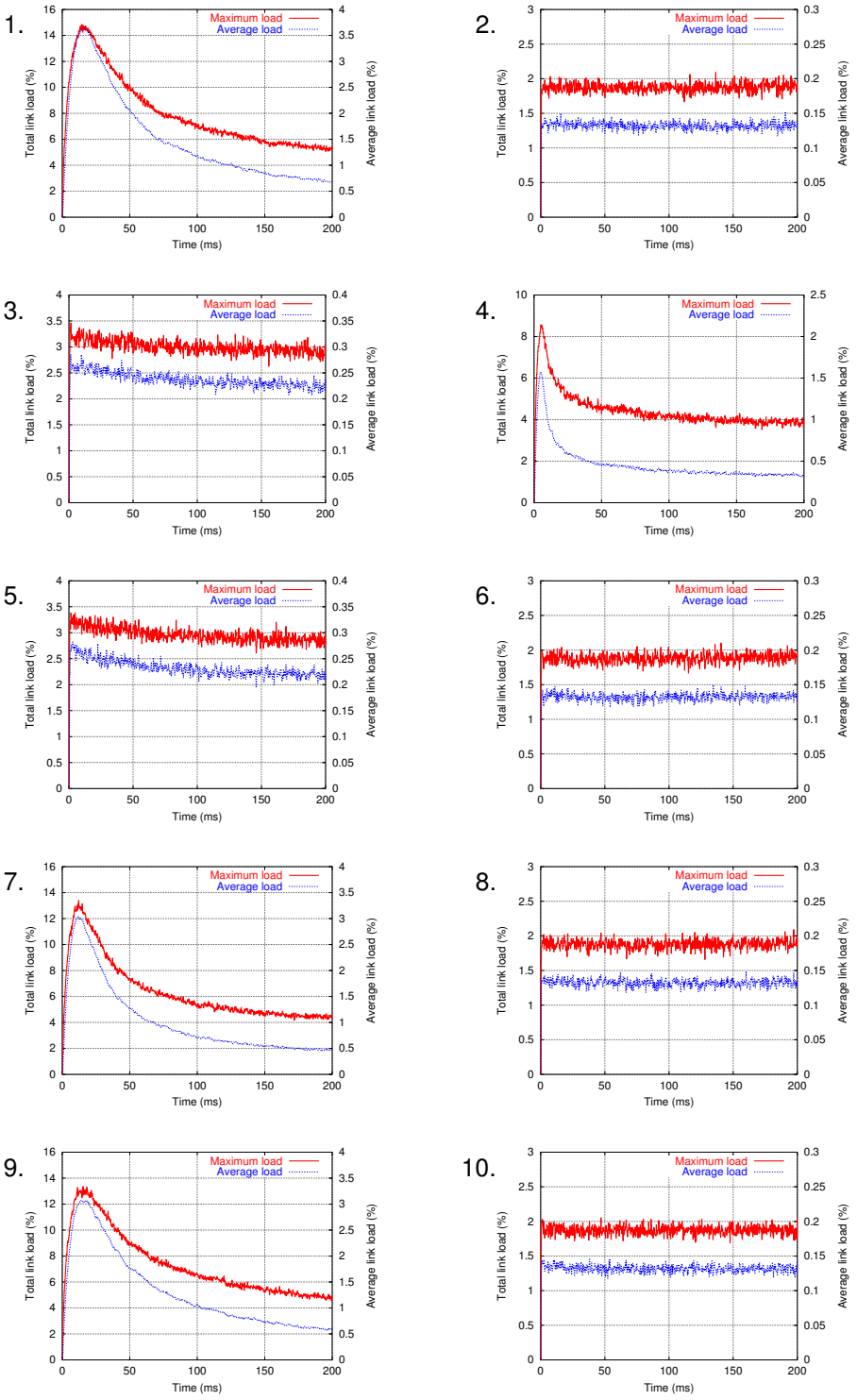


Figure D.4 Maximum and average load induced by the artificial ants using the **tree** topology.

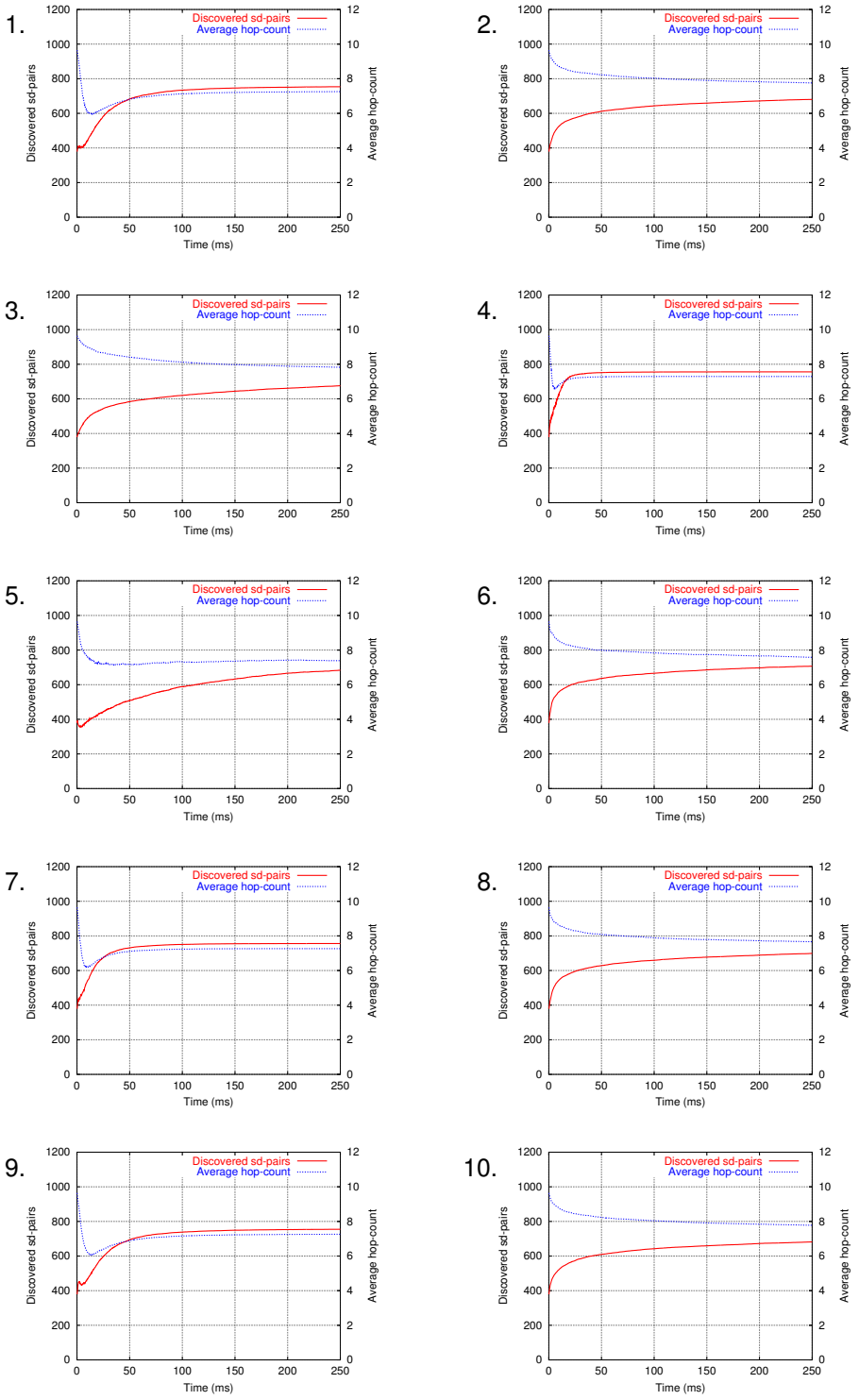


Figure D.5 The number of discovered routes between s-d pair and the average hop-count of the discovered routes using the **ring** topology.

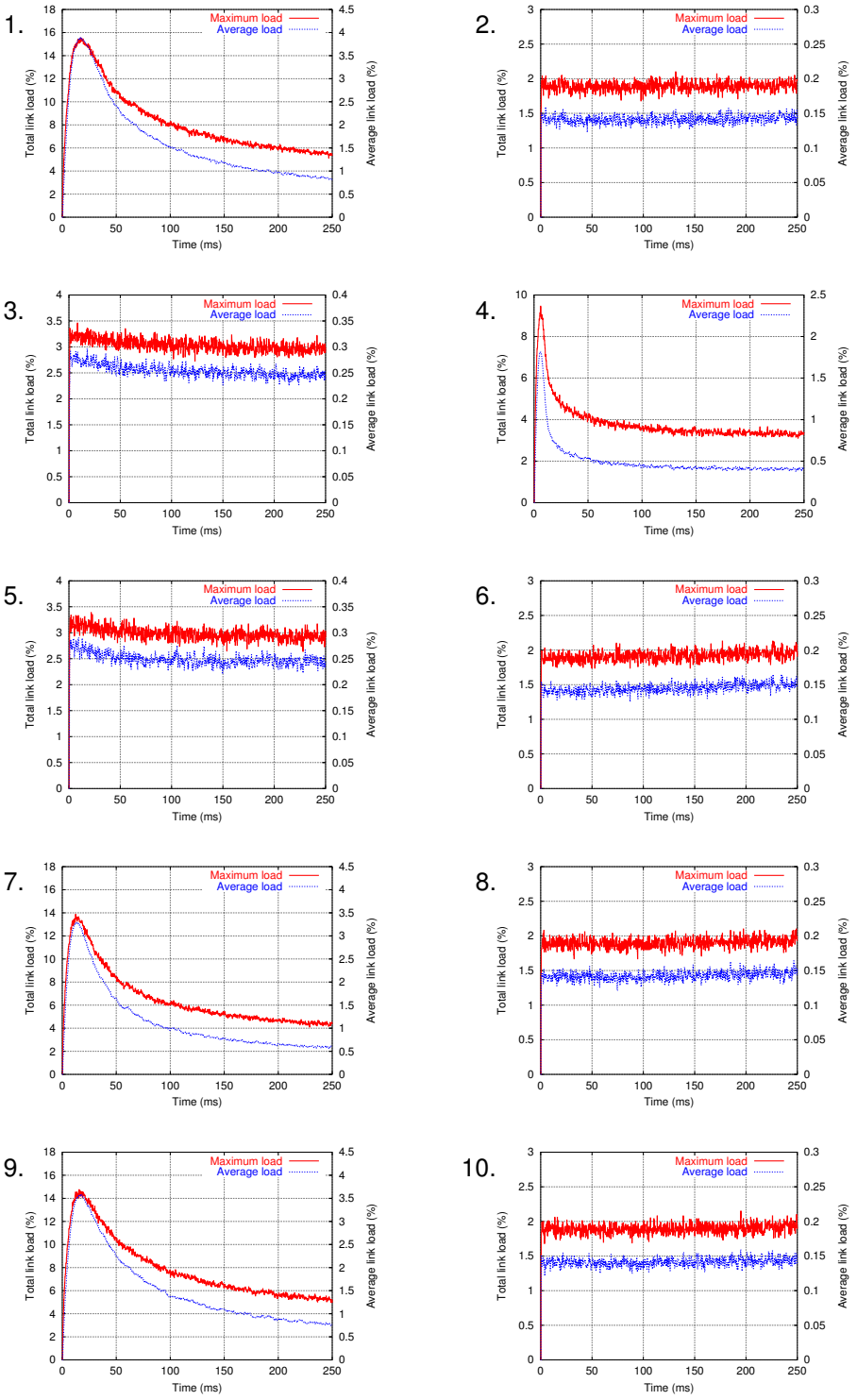


Figure D.6 Maximum and average load induced by the artificial ants using the **ring** topology.

D.3 Recovery from link failure

D.3.1 Failing links are not detected by the switches

The Figures D.7–D.10 (pp. 193–195) show how quickly the artificial ants recover from multiple links failures — i.e. how quickly the number of discovered routes get back to the optimum (756 routes). In all cases, all links are initially working, and at $t = 200$ms, one or many links suddenly stop working. These simulations have been made using the modes 1, 4, 7, and 9.

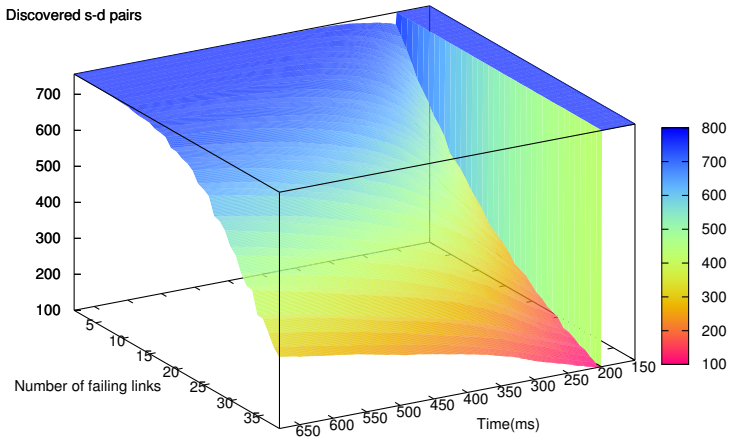


Figure D.7 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 1**.

D.3.2 Failing links are quickly dropped

The Figures D.11–D.14 show the number of known routes between all possible s-d pairs as a function of time and the number failing links. In these cases the switches *detect* that a link is not working and immediately the pheromone values for the failing port is set to zero. The old value is split evenly among the working ports. Once again, the simulations have been made using the modes 1, 4, 7, and 9.

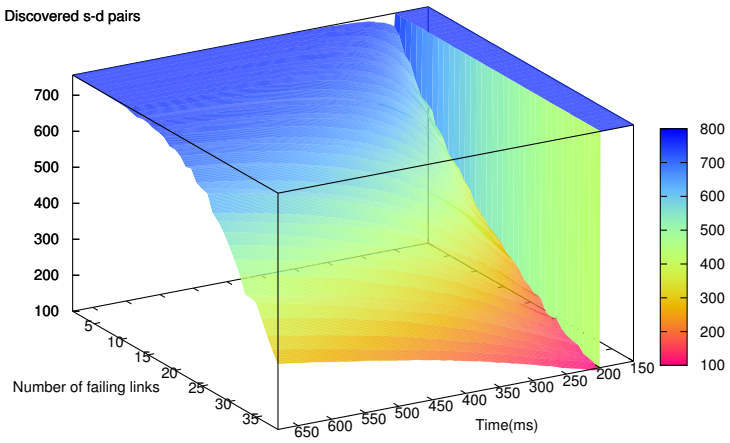


Figure D.8 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 4**.

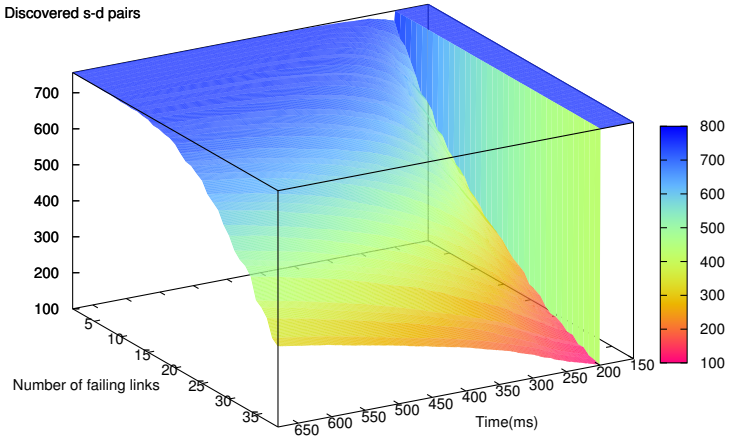


Figure D.9 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 7**.

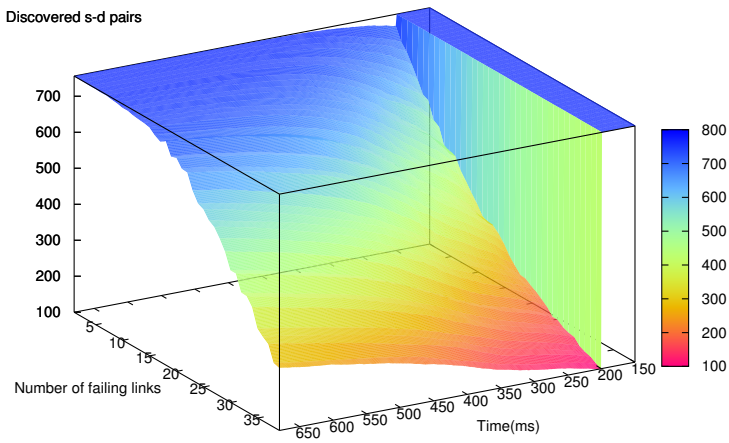


Figure D.10 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 9**.

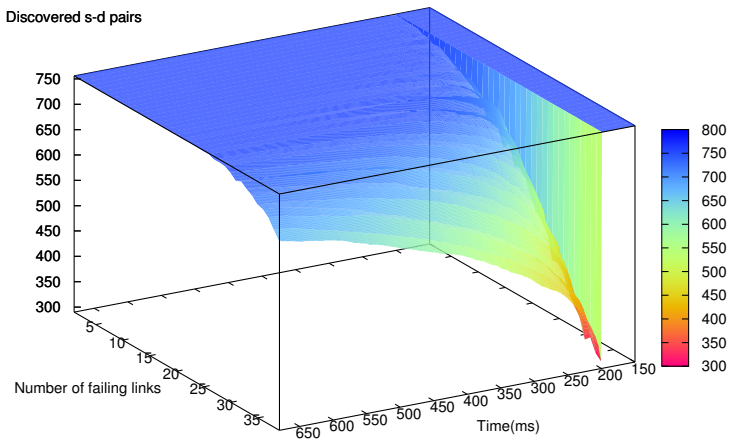


Figure D.11 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 1** and a mechanism to quickly drop broken links.

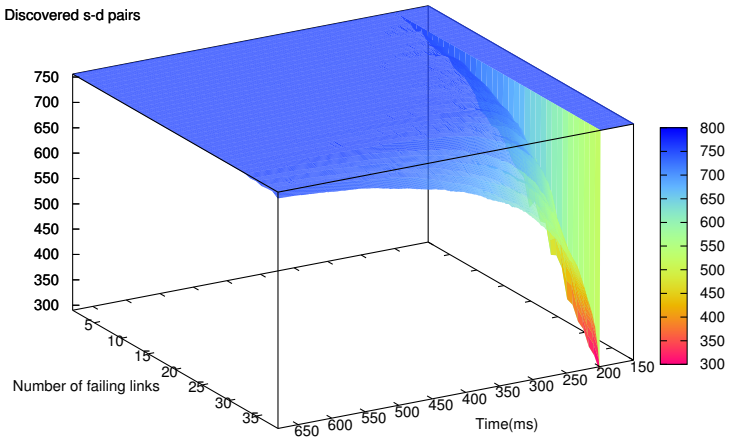


Figure D.12 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 4** and a mechanism to quickly drop broken links.

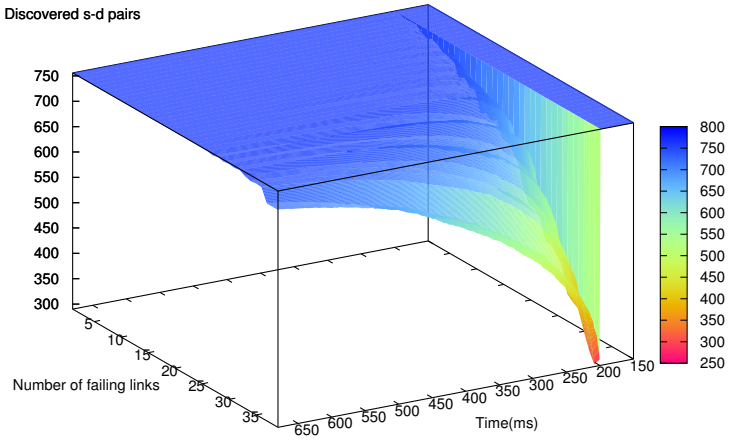


Figure D.13 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 7** and a mechanism to quickly drop broken links.

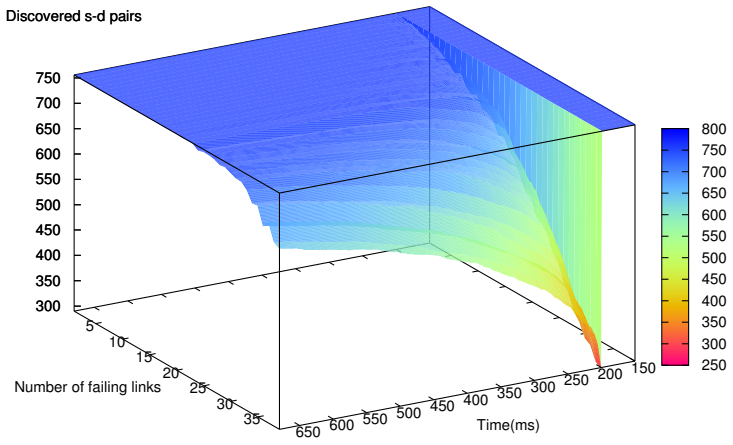


Figure D.14 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 9** and a mechanism to quickly drop broken links.

D.3.3 Failing links are quickly dropped (enhanced)

The Figures D.15–D.16 show the number of known routes between all possible s-d pairs as a function of time and the number failing links. As in the previous section, the switches *detect* that a link is not working and immediately the pheromone values for the failing port is set to zero. This time the old value is *not* split evenly among the working ports — instead it is split proportionally to the pheromone values of the other ports. Also this time, the simulations have been made using the modes 1, 4, 7, and 9.

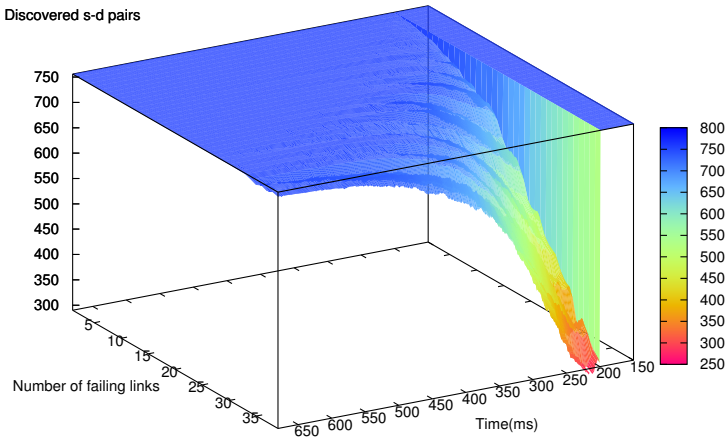


Figure D.15 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 1** and an enhanced mechanism to quickly drop broken links.

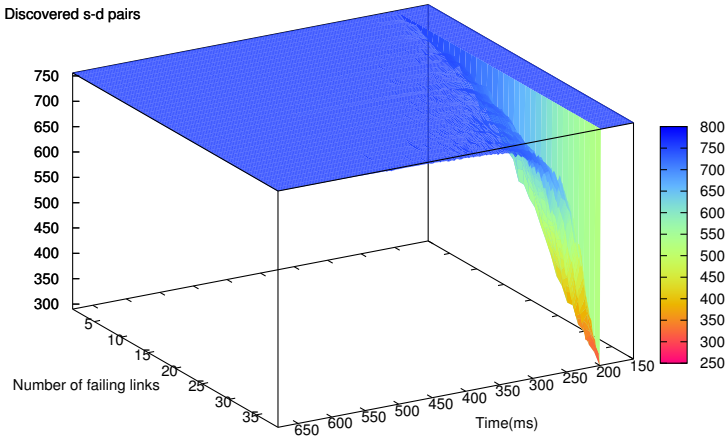


Figure D.16 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 4** and an enhanced mechanism to quickly drop broken links.

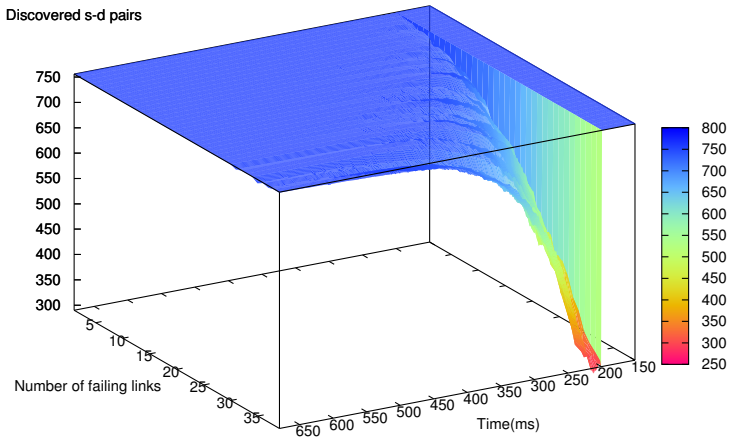


Figure D.17 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 7** and an enhanced mechanism to quickly drop broken links.

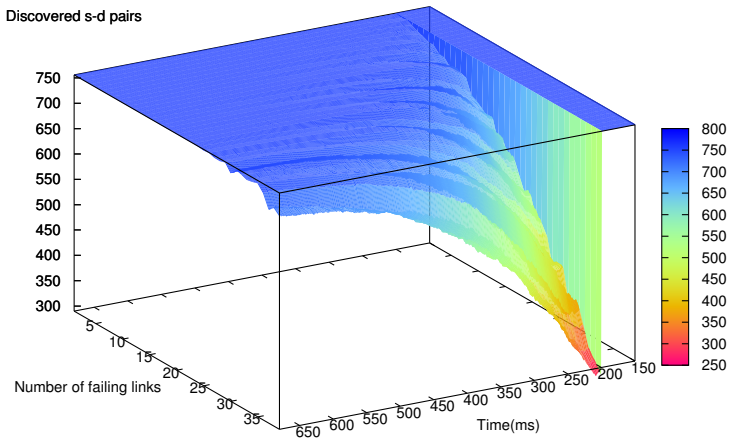


Figure D.18 The number of routes between s-d pairs as a function of time and the number of failing links using **mode 9** and an enhanced mechanism to quickly drop broken links.

D.4 Failing links are up and running again

The Figures D.19–D.30 show what happens after failing links are restored and ready to be used again. For this set of simulations, the system is in balance a $t = 0$ms (i.e. all routes are discovered, and the average length is near-optimal) and at $t = 10$ms a number of links are (re-)enabled.

When *many* links (10–15 or more) recover simultaneously a few of the 756 possible routes may get lost temporarily. However, after approximately 2 seconds the system has returned into balance. When links are disabled the average hop-count of the routes will, of course, be larger than when all links are enabled. Also, after a few seconds the average hop-count approaches the optimum — i.e. the recovered links are in use again.

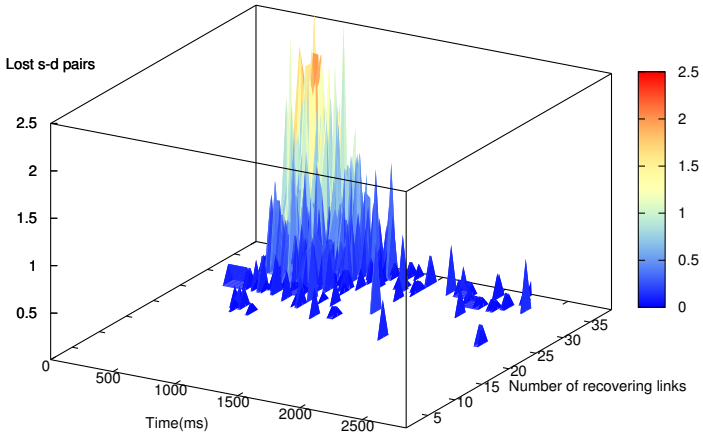


Figure D.19 Number of lost routes as a function of time and the number of links being enabled after the links have recovered (**mode 1**).

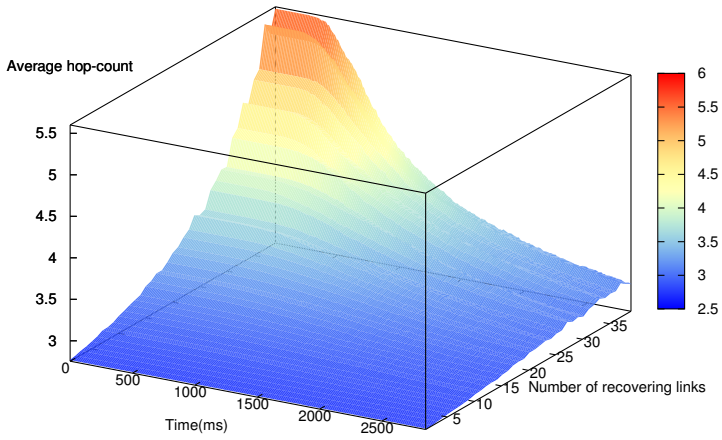


Figure D.20 Average hop-count of the routes after link recovery (**mode 1**).

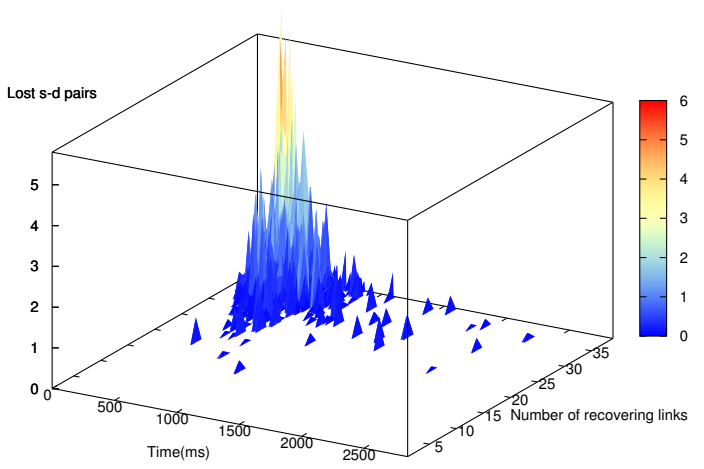


Figure D.21 Number of lost routes as a function of time and the number of links being enabled after the links have recovered (**mode 4**).

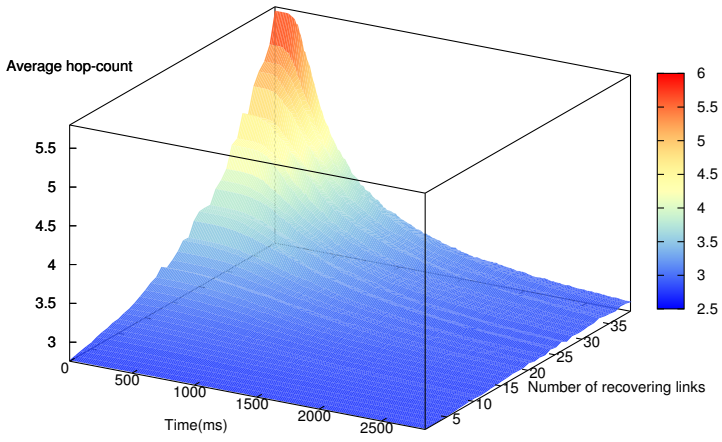


Figure D.22 Average hop-count of the routes after link recovery (**mode 4**).

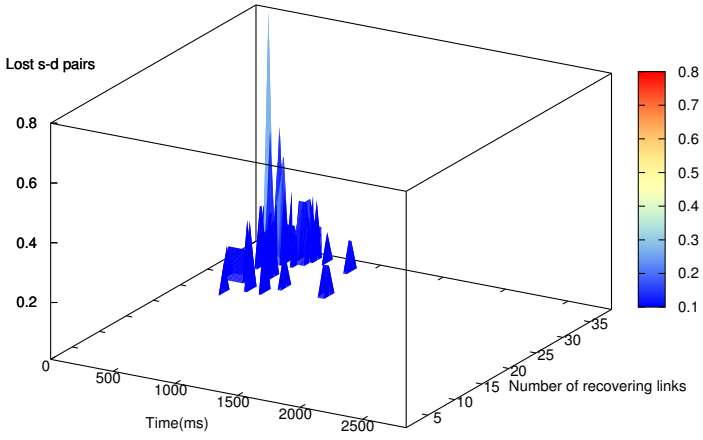


Figure D.23 Number of lost routes as a function of time and the number of links being enabled after the links have recovered (**mode 5**).

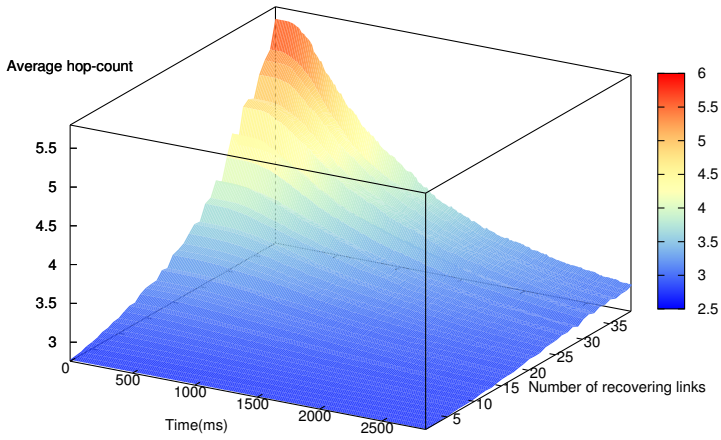


Figure D.24 Average hop-count of the routes after link recovery (**mode 5**).

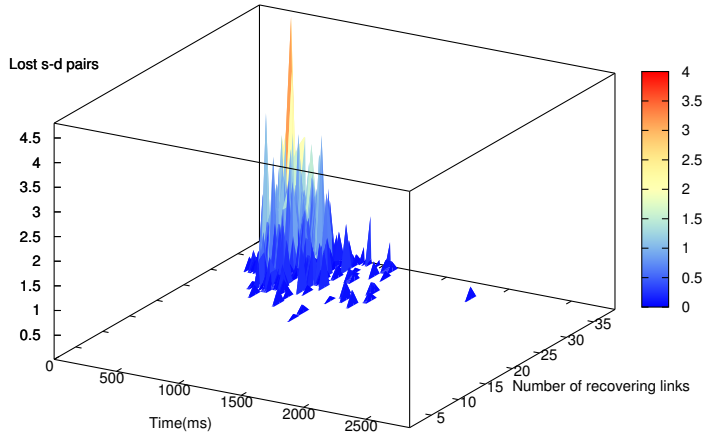


Figure D.25 Number of lost routes as a function of time and the number of links being enabled after the links have recovered (**mode 7**).

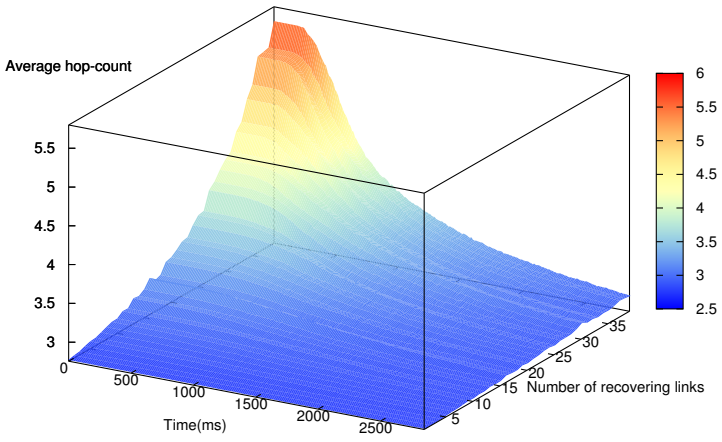


Figure D.26 Average hop-count of the routes after link recovery (**mode 7**).

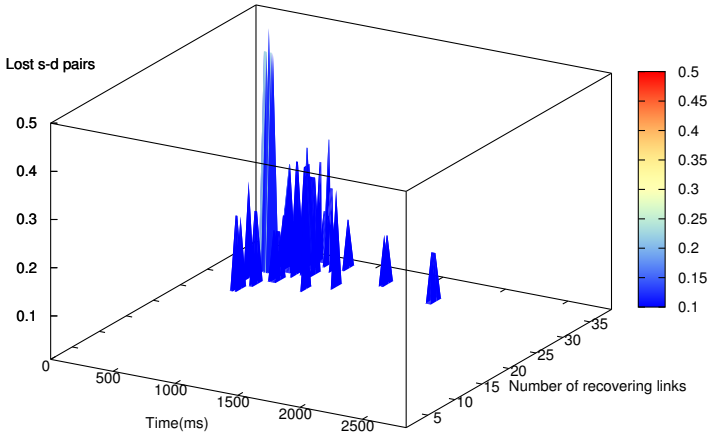


Figure D.27 Number of lost routes as a function of time and the number of links being enabled after the links have recovered (**mode 8**).

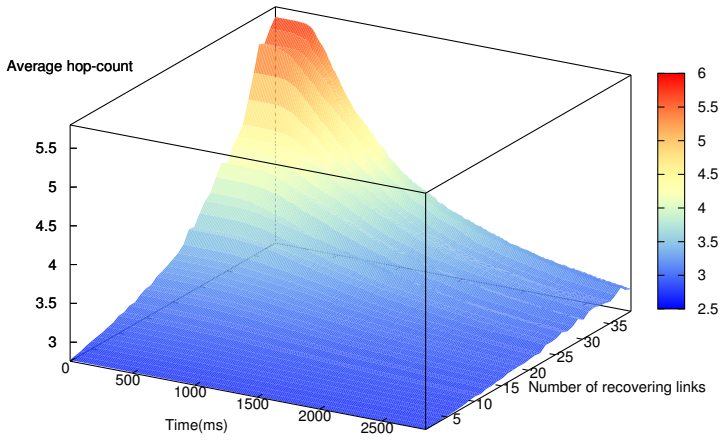


Figure D.28 Average hop-count of the routes after link recovery (**mode 8**).

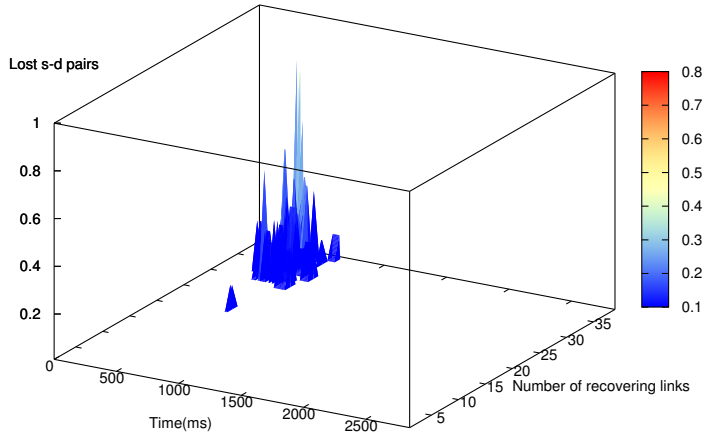


Figure D.29 Number of lost routes as a function of time and the number of links being enabled after the links have recovered (**mode 9**).

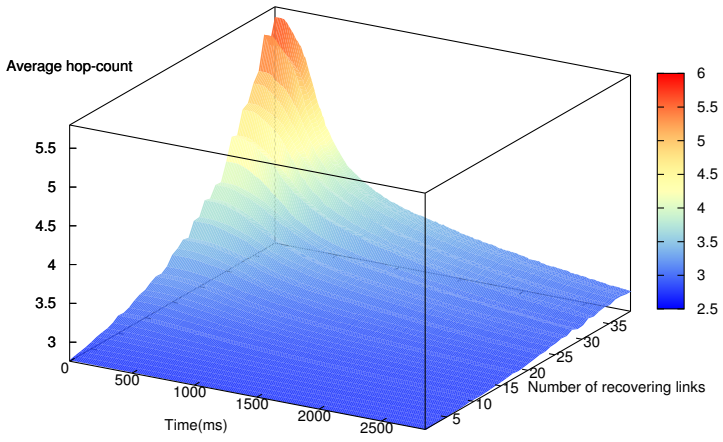


Figure D.30 Average hop-count of the routes after link recovery (**mode 9**).

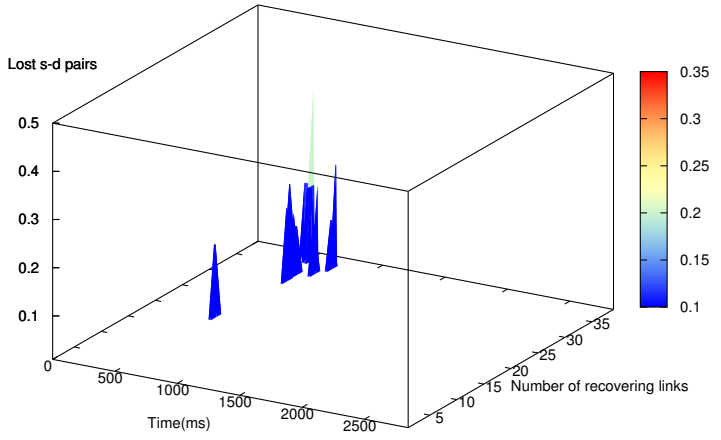


Figure D.31 Number of lost routes as a function of time and the number of links being enabled after the links have recovered (**mode 10**).

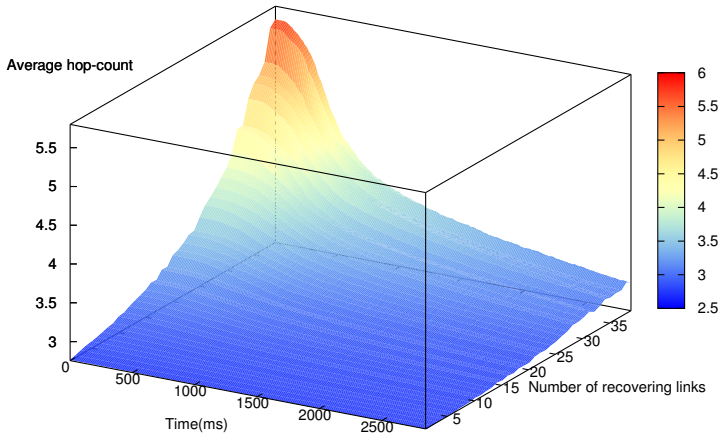


Figure D.32 Average hop-count of the routes after link recovery (**mode 10**).

Appendix E

Ant Simulator User Guide

This appendix describes the use of the ant simulator GUI. Although the GUI does not give access to all features of the ant simulator, it is still very useful for making oneself acquainted with the general behaviour of artificial ants. In the project the GUI has been particularly useful for running some quick tests before a batch job lasting many hours has been started.

E.1 How to obtain the simulator

The ant simulator is written entirely using Sun's Java SDK version 1.4. It does not work with older versions of Java or with Microsoft's Java implementation (which does actually not pass the Java compliance tests). The simulator can be run in two modes:

1. As an applet running inside the web browser.
    - Requires that Sun's Java runtime environment is installed as a plug-in.
    - Currently the applet version of the simulator can only be used with a small demonstration network.
    - Web-page: `http://stenhuus.dk/ants/demo/`
2. As a stand-alone application.
    - Requires that Sun's Java runtime environment is installed on the system — but not necessarily as a plug-in.
    - The network being simulated can be loaded from a file (see Section E.4).
    - The Java archive file containing the simulator is available at:
        - `http://stenhuus.dk/ants/demo/`

E.2 Starting the simulator

If the program is run as an applet within the web browser, it starts automatically when the web page is loaded.

Running the program as a stand-alone application is done as follows:

```
java -jar NetworkJApplet.jar network-file [plaf]
```

`network-file` is replaced with the name of the file containing the network that is going to be used for the simulation. The word `demo` is reserved for the built-in demonstration network used by the applet version of the simulator.

`plaf` is an optional parameter. PLAF is short for Pluggable Look-and-Feel, which is a Java SWING[1] feature that allowing the user to change to look-and-feel of the GUI. `plaf` can be one of the following:

`motif` is a UNIX-like theme (**default**)

`metal` is Java's own theme

`win` is the well-known Windows-like theme (this PLAF mode only works when it is run under Windows)

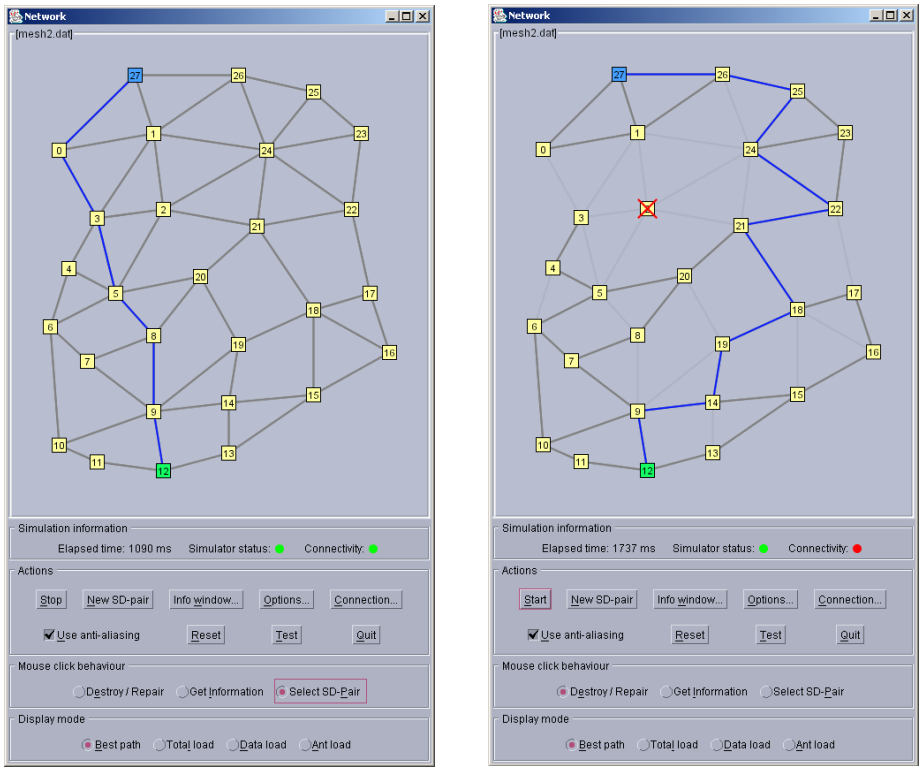
E.3 Using the simulator

The simulation is started simply by clicking the "Start" button[2]. Figure E.1 shows two screenshots of the main window. This window contains a graphical view of the network as well as some checkboxes and buttons. In Figure E.1a the GUI shows the path found by the artificial ants between the nodes 12 and 27. In Figure E.1b a part of the network has been destroyed, so the best path between the nodes 12 and 27 has changed.

It is also possible to get a graphical view of the data load and the load induced by the artificial ants. This is shown in Figure E.2.

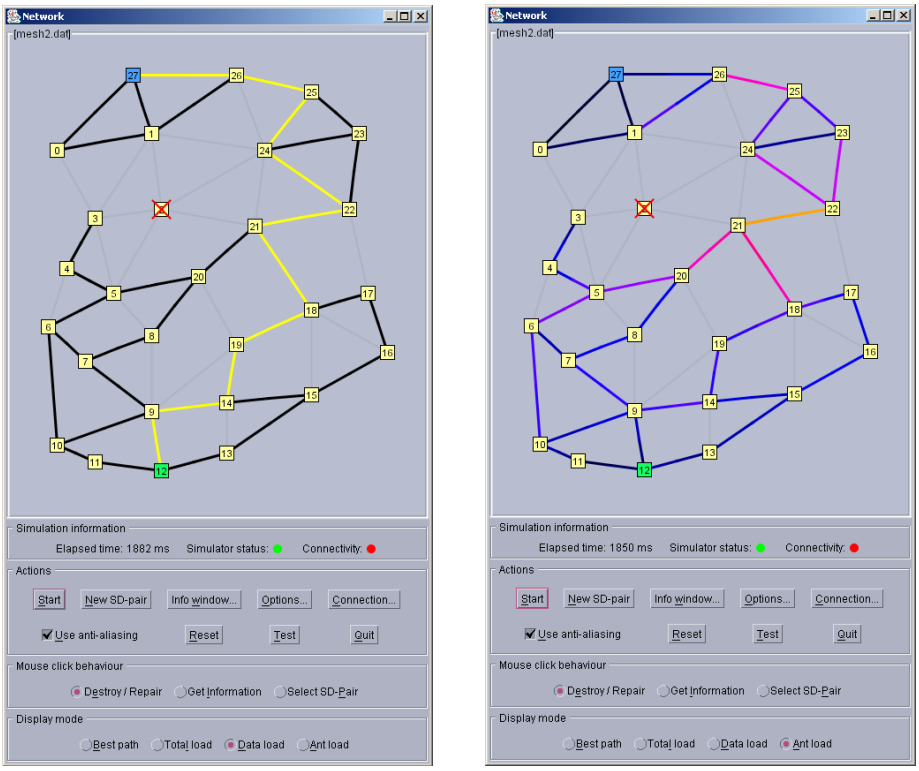
[1] SWING is a collection of libraries providing GUI primitives (widgets), a HTML-parser and more.

[2] The text on the button changes to "Stop" once the simulation is running.



(a) (b)

Figure E.1 Screenshot of the simulator's main window showing the currently best path between the nodes 12 and 27 when, (a) all the equipment is working, and (b) parts of the network has been destroyed.



(a) (b)

Figure E.2 (a) Screenshot showing the data load after one bidirectional connection has been established between the nodes 12 and 27. (b) Screenshot showing the ant-induced load of all links (the ant birth-rate has been set unrealistically high in this example.).

E.3.1 The "information" panel

The information panel is located directory below the network view inside the main window of the simulator. The information panel presents the following information to the user:

Elapsed time is the simulated time in milliseconds.

Simulator status is normally green (i.e. no problems). If it changes to red, a problem has occurred in the simulator (of course, this should never happen).

Connection status illustrates how many routes that have been found by the ants. It may have the following colours:

Red The number of discovered routes is below 99%.

Yellow The number of discovered routes is a least 99% but less than 100%.

Green All routes have been discovered.

Figure E.3 shows an example where 1090 ms have elapsed and both the simulator status and the connection status are okay.

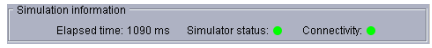


Figure E.3 Screenshot of the information panel (part of the main window).

E.3.2 The "actions" panel

The actions panel contains some buttons controlling different kinds of actions. Figure E.4 shows the actions panel.

Figure E.4 Screenshot of the actions panel (part of the main window).

The functions of the different buttons are explained in the following:

[Start/Stop] Use this button to start or stop the simulation. When the simulator is stopped it may be continued by clicking this button again.

[New SD-pair] When this button is clicked a new sd-pair is chosen randomly. The selected source gets the colour green and the destination is coloured in blue.

[Info Window...] Click this button to open (or close) the information window (see also Section E.3.5).

[Options...] When this button is clicked a window is opened (or closed) where the user can change the simulation options and parameters.

[Connection...] When this button is clicked a window is opened (or closed) where the user can add, edit, or remove connections.

[Use anti-aliasing] This checkbox can be set in order to use anti-aliasing in the network view. Anti-aliasing is normally preferred since it is more pleasing to the eye, however on slow computers setting this checkbox will have a significant impact on the performance.

[Reset] When the reset button is pressed, the time is reset to zero and the pheromone tables are put in their initial state. Pressing the reset button does not affect live connections and it does not remove data or ants from the links or buffers.

[Test] The behaviour of the test button may change between versions of the simulator — i.e. when pressing it the unexpected may happen. It has normally been used during the software development as a way to trig certain events or to test newly implemented methods. However, *normally* some information about the simulation is written to the standard output (STDOUT) the the test button is clicked. This could include things like the number of ants created so far, the number of discovered paths, the average hop-count of the discovered paths, etc.

[Quit] This button only exists when the simulator is run in stand-alone mode. Press this button to quit the simulator.

E.3.3 The "mouse click behaviour" panel

The "mouse click behaviour" panel is shown in Figure E.5

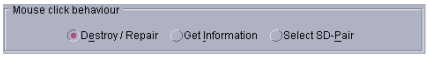


Figure E.5 Screenshot of the "mouse click behaviour" panel (part of the main window).

As the name of the panel suggests, the effect of clicking the mouse somewhere inside the graphical view of the network is controlled in this panel (the three behaviours are mutually exclusive). The meanings of the three different modes are:

[Destroy/Repair] When this option is selected, links and switches can be destroyed by simply pointing at them and clicking the left mouse button. If the clicked link/switch is already destroyed, a mouse-click will instead repair it.

[Get Information] In this mode, clicking a switch or link will cause some information about that switch/link to be shown in the information window (see Section E.3.5). If the information window is closed, the behaviour is still the same, and the information can be seen by opening the information window by pushing the "Info Window" button.

[Select SD-Pair] In this mode, an sd-pair can be selected by clicking two switches. The lastly clicked switch will turn green (i.e. it is the source) and the switch that was previously the source, will turn blue — i.e. it becomes the destination switch.

E.3.4 The "display mode" panel

The things that are actually shown in the graphical view of the network can be controlled in the display mode panel — see Figure E.6.

Figure E.6 Screenshot of the "display mode" panel (part of the main window).

The following four display modes exist:

[Best path] shows the best path between two nodes.

[Total load] shows the total load generated by data traffic and artificial ants.

[Data load] shows the data load only.

[Ant load] shows the ant-induced load only. Normally, the ant birthrate has to be increased from the default value of 0.001 in order to see anything interesting.

E.3.5 The information window

The information window can show some information about a single switch or link in the network. Figure E.7 shows the contents of the information window after the user has clicked the switch with the name "sw27" ("Get Information" has to be selected in the "mouse click behaviour" panel). The window shows the current values in the pheromone table of the selected switch, the buffer fillings, as well as the virtual connections with this switch as the source node.

If a link is clicked while in "Get Information" mode, the information window shows the following:

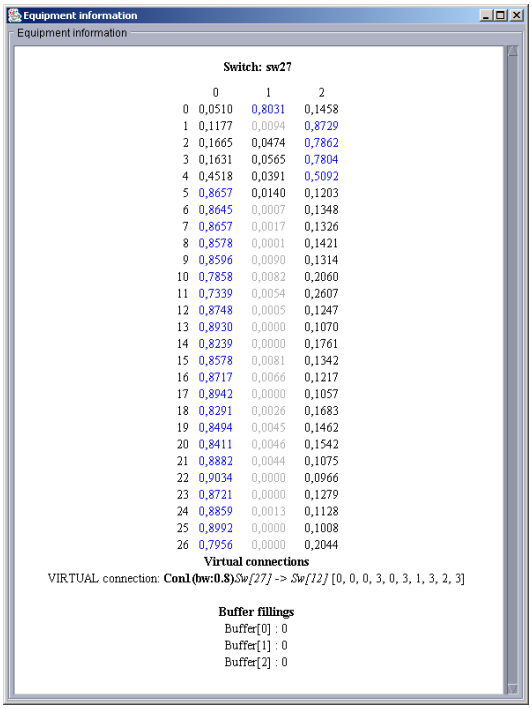


Figure E.7 Window showing the pheromone table of switch 27 as well as one virtual connection and the filling of the buffers (they are empty).

- The total load (max. value is 1000) and the total load in percent.
- The data load.
- The ant-induced load.

An example of this is shown in Figure E.8.

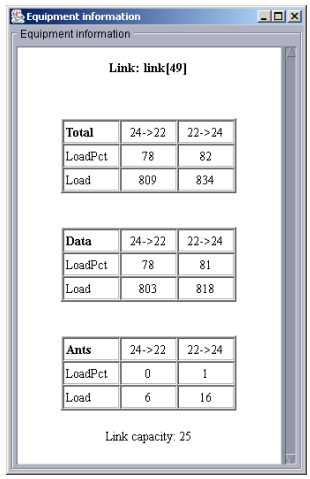


Figure E.8 Window showing the total load, the data load, and the ant load of one particular link ("link49").

E.3.6 Changing simulation options and parameters

A lot of options and parameters can be changed in the simulator. By clicking the "Options" button a window like the one shown in Figure E.9 pops up.

The following parameters can be set:

Initial feedback and Minimum feedback These two parameters decide the level of influence of each individual ant. If the values become too high, the system become unstable, due to the random nature of ants.

Noise level This number specifies the fraction of ants that will not obey the values of the pheromone tables. A certain level of noise is desirable to avoid dead-lock.

Ant birth rate This number specifies the probability that the next packet leaving a port will be a newly created ant. If the checkbox just below is cleared, the probability is per switch instead of per port.

Durability The durability specifies how enduring an ant is — i.e. this values determines the speed of ageing.

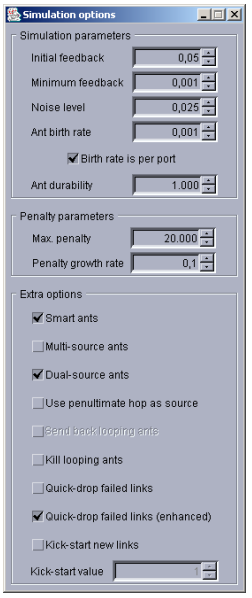


Figure E.9 In this window, options can be (de-)selected and parameters can be altered.

Max. penalty and Penalty growth rate The parameters control the penalty level as a function of the link load.

And the following options can be set (some options are mutually exclusive):

Smart ants This class of ants remember all nodes they have visited as well as the time, when they were at each node.

Multi-source ants When setting this option, smart-ants are treated at if they had multiple sources.

Dual-source ants When setting this option, smart-ants are treated at if they had two sources.

Use penultimate hop as source causes the second virtual source to be the node just visited.

Send back looping ants causes looping ants to be sent back to their source (where they are killed). On their way back, they try to erase their footprints.

Kill looping ants simply causes looping ants to be killed once they are disclosed.

Quick-drop failed links is a simple mechanism to remove ants from failed links.

Quick-drop failed links (enhanced) is a slightly better mechanism to remove ants from failed links.

Kick-start new links causes ants to use recovering links more quickly. This option should be used with care, otherwise loops may occur.

E.3.7 Adding and removing virtual connections

Figure E.10 shows the window where connections may be added or removed.

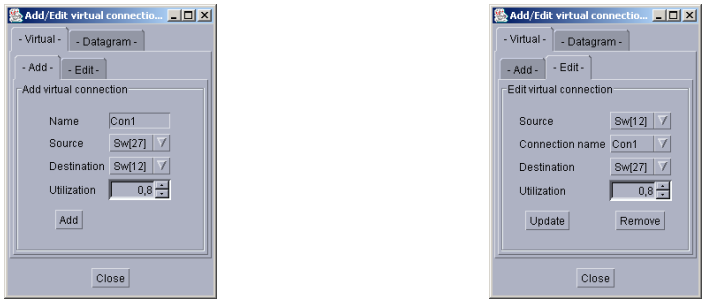


Figure E.10 Screenshot of the window where bidirectional virtual connections can be (a) set up, and (b) removed or changed.

In Figure E.10a a virtual connection called “Con1” is added between the nodes 27 and 12. The bandwidth of the has been set to 0.8, which means that the connection consumes 80% of the link load. When setting up virtual connections in this way, they will alway be symmetrical.

If the user wishes to remove a connection, this is done by first selecting to source switch, then selecting the connection name, and finally by clicking the “Remove” button. Figure E.10b shows an example of this.

E.4 Network text file format

The simulator uses a simple text-based file format to store network configurations. An example of a network with four switches and six links is shown below:

```
################################################
# Filename:  simple.dat
################################################
sw0  b25  e5  x1  y3
sw1  b50  e7  x1  y9  dis
sw2       e4  x9  y0
sw3           x9  y9

link0  l20  sw0  sw1
link1  l40  sw1  sw2
link2  l80  sw3  sw2  f4  dis
link3  l20  sw0  sw3
link4  l80  sw0  sw2  f4
link5  l40  sw1  sw3
################################################
```

Lines starting with a hash-mark (#) are comments, lines starting with "sw" each define a new switch, and lines starting with "link" define a new link. Figure E.11 show the network based on the shown example. The grey lines are disabled links — "sw1" is disabled, so all adjacent links do not work. Furthermore, the thick lines indicate that the speed of those links are higher than the speed of the other links.

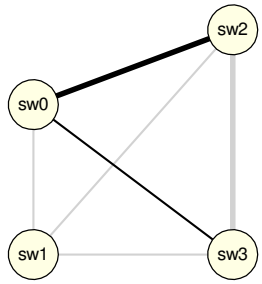


Figure E.11 Network based on network file format example.

E.4.1 The switch format

Creating a switch is done by adding a new line starting with the letters "sw". Between the first token and the end of the line a number of parameters may be specified. The following parameters are supported (none of them are mandatory):

b<integer> sets the size (in cells/ants/packets) of the output buffers of the switch (default value: 1000).

e<integer> sets the number of so-called slots within a switch (default value: 10). If a switch contains bursty data sources the number of slots should be large.

x<integer> is the x-coordinate of the switch used for the graphical view (default value: 100).

y<integer> is the y-coordinate of the switch used for the graphical view (default value: 100).

dis If this option is used, the switch is disabled by default.

A switch must be connected to at least one link, otherwise the file can not be parsed.

E.4.2 The link format

Just like switches, links are created by adding a line starting with the letters "link". The rest of the line is reserved for extra parameters. The following parameters are supported (the "sw" parameter is mandatory):

l<integer> set the length of the link (default value: 25). The length is specified as the maximum number of cells/ants/packets contained by the link at any given time. The default values of 25 corresponds to approximately 13 km optical fibre, if the link speed the 155 Mbit/s and the used technology is ATM.

sw<integer> This parameter specifies the switches to which the link is connected. A link *must* be connected to two switches.

f<integer> This parameter is the so-called speed-factor (default value: 1). The speed factor specifies how many time this link is faster than links with the default speed factor value.

dis If this option is used, the link is disabled by default.